(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 August 2001 (16.08.2001)

PCT

(10) International Publication Number
WO 01/59613 A2

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/IB01/00369

(22) International Filing Date: 9 February 2001 (09.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/182,070 11 February 2000 (11.02.2000) US

(71) Applicant *(for all designated States except US)*: **DATACHEST.COM, INC.** [CA/CA]; 442 St. Gabriel Street, Suite 100, Montreal, Quebec H2Y 2Z9 (CA).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **LAWEE, Alan** [CA/CA]; 272 Harrow Crescent, Hampstead, Quebec H3X 3X6 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/59613 A2

(54) Title: SYSTEM FOR DATA MANAGEMENT

(57) Abstract: A system and method for classifying, organizing and integrating raw information of all types into a form that permits the retrieval and analysis of the information by means of a simple user interface. Once integrated into the database and classified into the polyhierarchical tree structure, it is possible to request information using relationship that did not previously exist in the source data.

SYSTEM FOR DATA MANAGEMENT

FIELD OF THE INVENTION

The present invention provides a means by which one can integrate data from a variety of databases each with its own content, organization and structure into a single repository. The user can then retrieve and display the integrated data in its original form.

BACKGROUND OF THE INVENTION

Others have tried to cross-index data across the original source data, but only achieved an ability to index the data. Retrieval was then left to the original source application. As well, this approach did little to normalize the data, and achieved only limited integration. Conventional wisdom frowned upon integrating the data into a single repository as too labor-intensive and difficult. As well, until recently, few technologies were available to manage databases of this size with the necessary speed and flexibility required by a general query program.

BACKGROUND OF THE ART

Several patents exist which may be relevant to the present invention. Such U.S. Patent Nos. are:

4205371
4908759
5426780
5450581
5515534
5566332
5596746
5600826
5721912
5740421
5764973
5778375
5787433
5940832
5999937
6014670
6016501

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention assists in classifying, organizing and integrating raw information of all types into a form that permits the retrieval and analysis of the information by means of a simple user interface. Once integrated into the database and classified into the poly-hierarchical tree structure, it is possible to request information using relationships that did not previously exist in the source data.

Once the data has been integrated into a repository specifically designed to be extensible (designed in a manner similar to a meta-data database) the problem of integrating disparate databases becomes considerably easier. Retrieval and classification of the data then becomes the next problem to be solved. To do this, a poly-hierarchical tree-structure was implemented, which provided a number of important and beneficial side effects. The poly-hierarchical tree ("PHT") is a data structure with branches and leaves. The branches describe the hierarchy (topics and sub-topics), and the leaves describe the information nodes. The 'poly' portion of the name refers to the fact that a node (either a branch or a leaf) can appear more than once in the hierarchy. So to use an example, the 'Toxicology' branch appears below, the HEALTH branch, as well as the CHEMICAL PROPERTIES branch, and in both locations, it leads to a sub-hierarchy of branches and leaves that are below it. The poly-hierarchical tree provides an inherent context to each data node that can be used to enhance and optimize text searches and other relationships to the data node.

The present invention can operated on various hardware configurations. A preferred component list is as follows. One Unix Server services the Oracle Database Engine with two database instances: a transactional database and a data repository database; a second server (Unix or NT) services the WebLogic Web Application Server. In a fault-tolerant production environment each Server is duplicated. The two WebLogic Servers are operated in load-balancing/fail-over mode, and are connected to one of the two Database Servers containing the Primary Transaction Database. The Secondary Transaction database is operated in a hot standby mode continually updated with the archive logs from the Primary Transaction Database. Both of the Data Repository Databases are connected in load-balancing/fail-over mode to the Primary Transaction Database, which passes along the data requests generated by the Web Application Server.

One can search across granulated data as related to the text/images. Text can be processed (parsed) to extract the information that allows it to be integrated into the database, i.e., the substance name, manufacturer, CAS number, Physical State, or the like. Images, on the other hand are preferably accompanied by an index containing the necessary information in order to be able to integrate them into the database.

The context features are used to refine a full-text search. For instance, if a user has descended to 'Toxicology' from the 'Health' node, there may be some peer nodes in its immediate vicinity relating to workplace standards, exposure limits, First Aid Indications, and the like. These neighbors provide a context with which to weight the results of a full-text search. They might also define a specific collection of information within which to search for results, excluding a more general search that would return unrelated documents or data. If the user had reached 'Toxicology' from the 'Chemical Properties' node, the neighbouring branches and leaves might weight the search towards other property information, such as, but not limited to, Reactivity, Radioactivity, or the like, which would result in a different set of results.

A novel aspect of the present invention is the use of a PHT type of a structure in conjunction with full-text indexing and retrieval techniques. It is believed that such use within an RDBMS context to classify and sort the meta-data (Leaf descriptions, or what would be referred to as columns in a traditional 2-dimensional data table) in the metabase has heretofore not been disclosed.

As well, the concept of designing the metabase to be able to receive and integrate data from disparate sources with undetermined formats and structures is also innovative. The integration

provides a possibility which was not present in the disparate databases taken alone: the ability to perform combinatorial searches across all of the disparate data taken together. Furthermore, these searches are not related to a single technology (i.e., either a relational (SQL) search or a full-text search, or a bit-mapped query) alone – they can now be combined and manipulated together.

The present invention also provides a process for integrating data from a variety of databases each with its own content, organization and structure into a single repository, whereby the user can then retrieve and display the integrated data in its original form, comprising:

a. transferring data from a source medium into a staging area;

b. converting said data into a database format;

c. mapping said data into a master Index Tree;

d. normalizing or de-normalizing at least a portion of said data into a format suitable for parsing and integration into a target database;

e. parsing said data to extract granular data;

f. exporting said data of step e. into said target database;

g. formatting said data in a normalized form;

h. verifying said target database to ensure integrity has been maintained and reproduction has been accurate;

i. archiving said target database to a storage media;

j. merging said data into a master database repository;

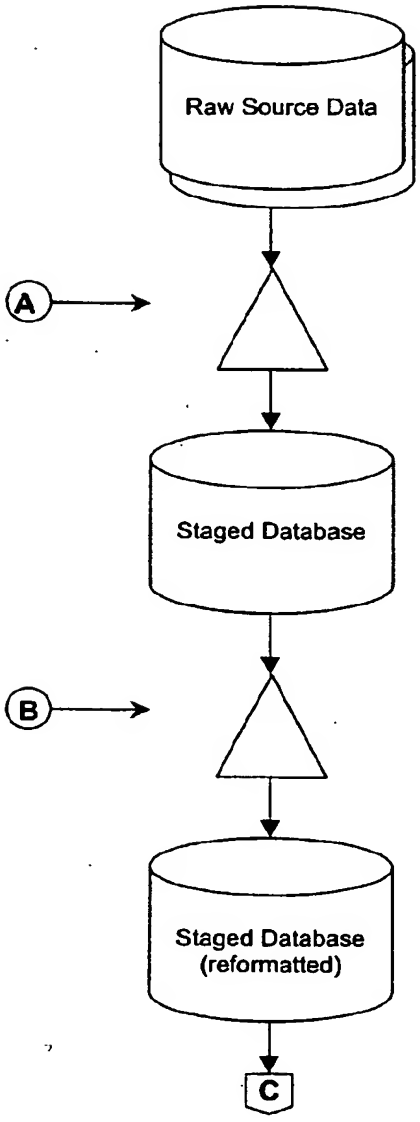
k. exporting said data from step j. to a pre-production instance; and,

l. de-normalizing said data from step k. so as to optimize access time in an online environment.

Additional disclosure material is attached hereto and incorporated herein.

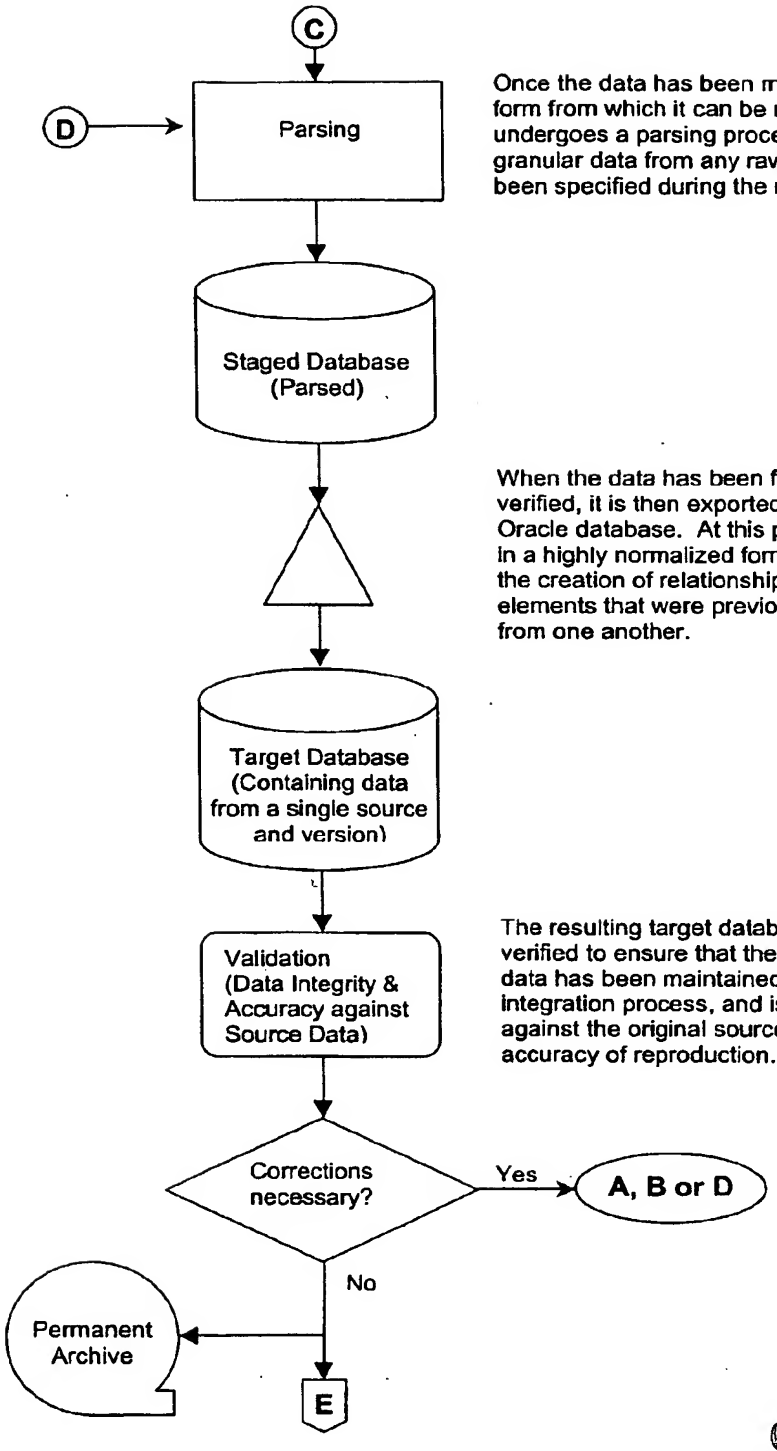
While the invention has been described in connection with certain preferred embodiments, it is not intended to limit the scope of the invention to the particular forms set forth, but, on the contrary, it is intended to cover such alternatives, modifications, and equivalents as may be included within the true spirit and scope of the invention as defined by the appended claims. All patents, applications and publications referred to herein are hereby incorporated by reference in their entirety.

EXPRESS MAIL LABEL NO. EL660499681US



The data is moved from the source media and imported into a staging area. Where appropriate, the data is also converted into a format suitable for, and then imported into a Microsoft® Access database.

The resulting data, in database format where appropriate, is then analyzed and mapped into the master Index Tree. Where necessary, the data is de-normalized or normalized into a format suitable for parsing and integration into the target database.

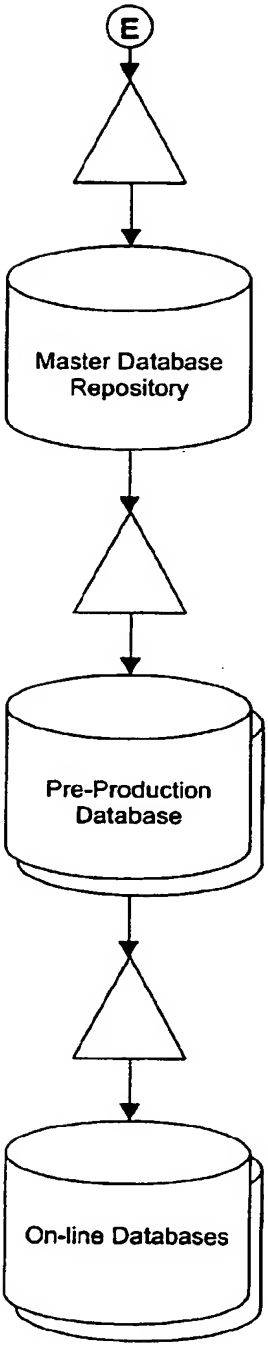


Once the data has been massaged into a form from which it can be manipulated, it undergoes a parsing process to extract granular data from any raw text that has been specified during the mapping process.

When the data has been fully prepared and verified, it is then exported into a target Oracle database. At this point it is formatted in a highly normalized form, which facilitates the creation of relationships between data elements that were previously independent from one another.

The resulting target database is then verified to ensure that the integrity of the data has been maintained throughout the integration process, and is also verified against the original source data to ensure accuracy of reproduction.

005



Once the target database is verified and approved, it is archived to a permanent media. The data is then merged into the master database repository.

The data in the Master Database is exported to a Pre-Production instance and de-normalized in order to optimize access time in an on-line environment.

The Pre-Production database is used as a final testing area and serves as a source database for replication to the various on-line databases.

006

Data Architecture1 clean.doc

Database Architecture

DataCHEST will be storing data from a wide variety of providers. Inevitably, each provider will have their own preferred structure for organizing their data. Our challenge is to design a database architecture that is flexible enough to accommodate the importing of data from all of the different provider databases. However, the architecture must also be capable of indexing the data so that it can easily be queried and retrieved by our search engine with fast response times and minimal bandwidth usage.

Two choices present themselves: to create a tailor-made, proprietary structure within which to store our providers' data, or to design a database architecture that can be implemented within a third-party database and Internet/Intranet Web environment.

Some of the commercially available database and Web applications that we are able to consider include :

- Microsoft SQL Server and Internet Information Server
- Oracle
- Informix
- Sybase and Power Tools
- Microsoft Access
- Microsoft FoxPro
- Dbase
- IBM's DB2
- Object-Oriented Databases (O_2, Objectivity, etc.)
- Atrion International's Chemmate® proprietary database engine
- Showbase Extra and Showbase Collaborative (http://www.showbase.com)
- Sand Technology's Nucleus Data Warehouse Server (http://www.sandtechnology.com)

A common thread runs through nearly all of the above databases. They are, for the most part, ODBC-compliant, and are able to use an SQL-like language to query and manipulate their data. Unfortunately, this ability works against the very flexibility that we are looking for in order to organize and store our data.

Among the exceptions in the above list is the proprietary database engine developed by Atrion for its Chemmate® software. Although the development of Chemmate® began in the early 1990's, when ODBC was not yet universally accepted, nor fully defined, a variety of commercial database applications existed at the time that could have been utilized. Atrion chose instead to develop their own engine, primarily because the nature of their data made it awkward to store in a traditional 2-dimensional relational database table.

The structure of the data required for Chemmate®, or more precisely the lack thereof, provided the impetus for the Atrion designers to embark on creating their own proprietary solution. Eventually, market pressure forced the company to develop an ODBC interface to enable the application to store its data within an Enterprise Database environment. Many parallels exist between that situation, and the one being faced by DataCHEST today. A close examination of their decision, especially in light of the tools available today, would be very instructive in our current situation.

Other solutions noted towards the end of the above list also employ a non-standard relational structure. The ShowBase Extra and Collaborative applications are able to import from ODBC-compliant data sources, but rely on their proprietary architecture for improved query and access speed. As these products are designed primarily for publishing data on the Internet from off-line data sources, this strategy is an acceptable one for our application.

The Sand Technology Nucleus application is a revolutionary new ODBC-compliant database engine with a unique, proprietary underlying structure, which was designed to optimize the storage of typical data in data warehousing environments. In a nutshell, it converts all values to binary tokens (somewhat akin to Chemmate® ELID's), creating tables which map tokens to their original values, and then creates a compressed database consisting only of tokens. The resulting compressed database is again transformed into binary matrices of value sets, which are again compressed. The binary nature of the resulting structure is much more efficient for searching, and consequently, Nucleus automatically indexes all fields in the table, nearly always without a speed penalty.

The present situation

The following statements characterize some of the different aspects of the data that will be stored in our database:

- There will be many (several thousand) different fields *(table columns)* relating to each substance. *(We intend to combine the data from all of our providers into a unified format, accessible from a single user-interface.)*
- There may often be more than one source and corresponding value for a data element. *(The contents of a cell can be multi-valued.)*
- Not all field values will be present for every record. *(The table will be sparsely populated.)*
- The format and size of the data contained in a table cell is indeterminate. It could be anything from a simple Boolean value (True/False), a numeric, or a text string, to a complex array of values, a bit-mapped image, a text document or a hyperlink.
- The databases are multi-lingual.
- The size of the database and the number of data elements will be extremely large.
- The data itself will be completely static and user access will be limited to read-only. Updates will occur off-line on a standby database, which will be toggled to an on-line state when the update is complete.

The data can be separated into two independent categories:

- Texts of law or policy (which may relate to classes of substances, but rarely to a single substance in particular).
- Substance-related data (Properties, presence on a list, MSDSs and other related documents, etc.)

At the moment, it is believed that the data architecture requirements for storing texts of law and policy are well served by existing full-text indexing technology, and will not require special study.

To speed up the initial product release, it would be acceptable to index substance-related data by only a few selected fields. These would include fields like chemical name, CAS number, EINECS number, ELINCS number, UN Number, IATA number, chemical formula, molecular structure, supplier name and part number, etc. In other words, only by fields that serve to identify a substance rather than those that characterize it.

This will permit users to search for desired substances and to retrieve desired data relating to those substances, which encompasses the bulk of the present demands on the data from the user population.

However, as the user population grows, it is believed that new uses and applications will require that all of the data fields be indexed. This would permit the application to rapidly handle queries like '*Retrieve all products whose flash point is above 30° C, whose specific gravity is less than 1.5 and which are not listed in EU regulations as R27 in concentrations greater than 14%*'.

For that reason, it is preferred to begin with a robust database environment that can provide the tools necessary for DataCHEST to develop a basic package quickly, but which can also support the development of a more sophisticated application over the medium term.

Data Model

A potential Data Model of relationships has been included below. It should be noted that the design is far from final, and that additions and changes will be made as we consult various experts, and as the project progresses.

As is evident from the figure, nearly all of the tables relate either to the Substance Table, which contains identification fields only (the ones shown are for example only, and more will be added), or to the Field Description table. Actual substance-related data is contained in a set of tables distinguished by data type. Data elements are uniquely identified by a substance, an Authority, a Language (where applicable), and a date of last update of the data element (when the element was updated, not when the source of the data was last updated).

A Field Table lists the various fields that have been defined for substance-related data. Each field has a unique data type, which is related to the data table in which the data is stored. For clarity, the Data Model is shown with four major data types (Boolean, Numeric, Text, and Unstructured), but additional types are also contemplated. Examples of these would be numeric ranges, hyperlinks, and other types of data.

Field Descriptions are organized into multi-level tree-structure of Groups for easier searching and selection.

An Authority Table lists the various Information Providers that supply the data in the database, together with the version and date of last update of their source database.

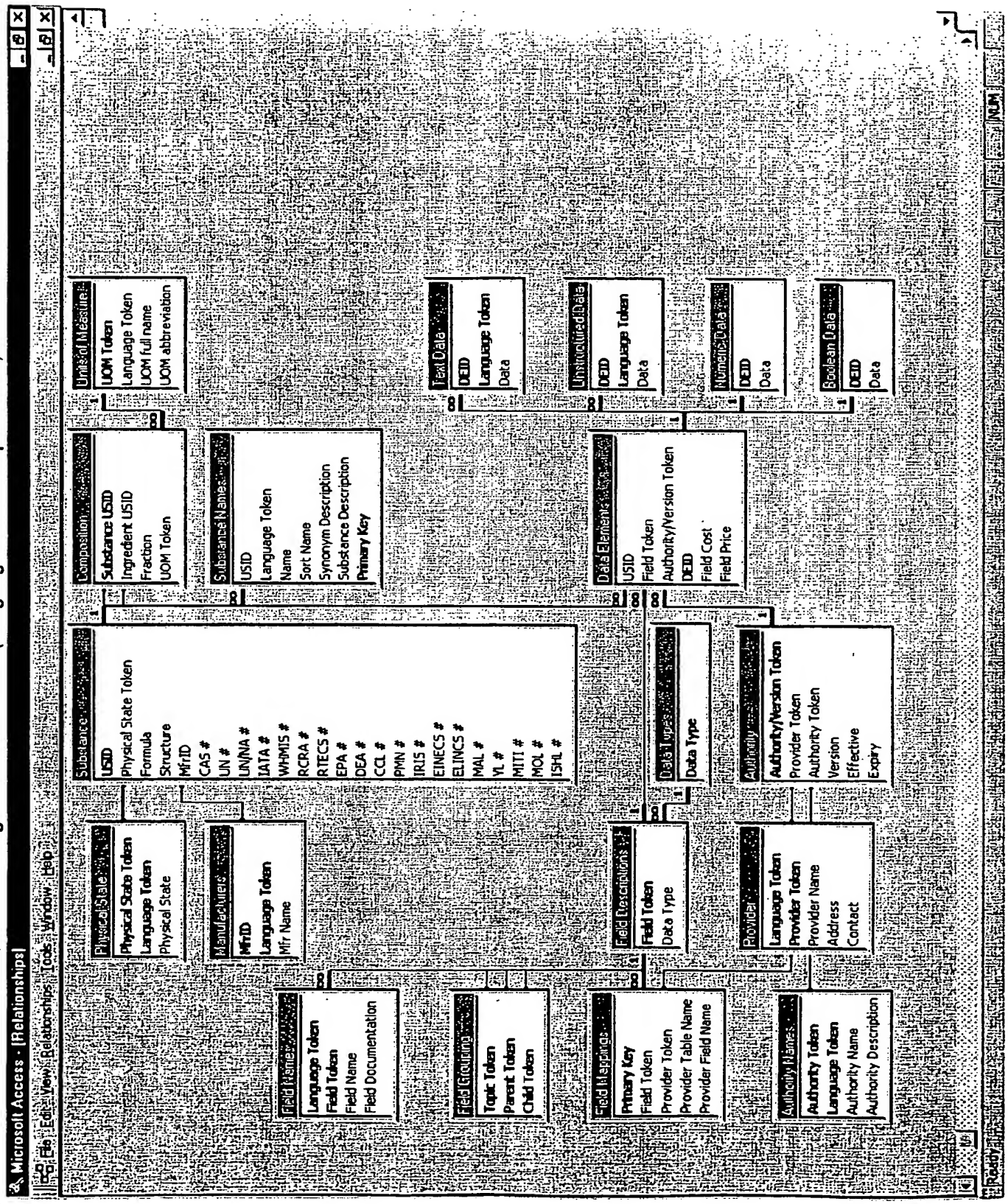
A Language Table lists the various supported languages and is keyed to those tables that are able to hold multilingual data.

A Field Mapping Table lists the mapping between Field Descriptions in our Database and the field description and source table in the Information Provider's native database.

A Synonym Table provides an important tool to facilitate searching for substances. Substance names are often vague, referring at times to a unique substance and at other times, referring to a class of substances. By allowing duplicate Synonyms, the Synonym Table structure will provide for this condition.

A Composition table provides a means by which it is possible to represent mixtures by listing each ingredient (which itself must exist in the Substance Table) together with its concentration. Note that the data will simply be imported from the Information Provider. No attempt will be made to ensure that the total ingredient concentration will add up to 100%.

Figure 1: Data Model (Language relationships hidden)



User Interface

Ideally, the user should have two modes in which to access the data:

- Interactively, with results displayed in HTML format in his browser window, or
- Enfielded, with results returned in an ODBC/JDBC-compatible table that can be stored in a local database or in a delimited flat file.

DataCHEST will promote a 'shopping list' approach for the user interface.

The initial choice will permit the selection between Documents (Texts of Law or Policy, bit-mapped images, etc.) and Substance-related Data. As indicated earlier, the document retrieval procedure will utilize off-the-shelf Full-text Indexing Search Engine and will be dealt with in a separate document.

The next steps for retrieving Substance-related data can be performed in any order, and will resemble the 'Wizard' approach common in the latest 32-bit applications on the market.

- The user must identify whether his query will be Interactive or Enfielded.
- The user must specify the product selection criteria (e.g. «All products whose name contains the text "DIMETHYL"», or «All matches for CAS # "[illegible]-00-1"», or simply «TRICHLOROBENZENE»).
- The user must specify which Data Elements (fields or properties) he wishes to retrieve. (e.g. Boiling point, Melting point, Flash point, Specific Gravity, European Annex 1 status and concentration limits, Presence on TSCA list and reportable quantities, etc.)
- In many cases, the user will have the option of specifying which Provider(s) should be consulted. This can be done at the time of the query, but an option will be provided to set up preferences, priorities and methods in advance. (e.g. «Consult ChemTox first, then LoLi, then NCEC» or «Take the average of the values provided by ChemTox, LoLi and NCEC» or even «Return the range across all of the values provided, excluding any elements that deviate from the mean by more than one standard deviation»)
- The user will also be able to filter the results according to the date of last update. This will permit restricting a search to exclude data that was already retrieved by a previous request, and/or requesting data from a specific period or version that was published by the Information Provider. (e.g. «Select data from ISIS Q4'1995, last updated between 9/15/1995 and 12/31/1995».)
- The user will be able to specify which Languages to retrieve for textual or language-specific data elements. (e.g. «Retrieve Japanese language MSDS documents for "Acetic Acid"».)
- Lastly, the user will have the option of requesting that the application calculate the fees to be charged for the transaction, and displaying them for approval prior to returning the request. In this case, the result set will be cached at the server for an appropriate period of time so that the query will not have to be repeated.

Once these steps have been completed, the query will be performed and the results returned in the desired format.

The ideal user interface would be a combination of a drill-down menu-driven structure, or Query Wizard, that would display the various options available at every step of the process, and a natural-language component that would allow a relatively flexible format for expressing a query.

Particular attention will have to be made to the definition & grouping of the Field Descriptions. The large number of potential field names will give rise to confusion. Rather than being related to the data architecture and systems design, this issue is actually an operational one. New fields will have to be defined for each database that is added, and these fields will have to have a common naming convention and be organized in common groups if they are to be compared with each other. Given that each Information Provider will have their own Field Description, we will have to be very disciplined in assigning our own.

Not to belabor an example, but, if a user is looking for Boiling Points, and the data is available from 5 different sources, but in varying formats, there may be 3 or four different Field Descriptions to describe the information. Consider, for example, the following data set:

Field Description	Authority	Last Updated	Value
Boiling Point, degrees Celsius	ISIS	98-01-01	23.25
Boiling Point, degrees Celsius	LoLi	98-07-01	23.3
Boiling Point, degrees Celsius	LoLi	97-10-01	23.2
Boiling Point, degrees Celsius, range, lower bound	NCEC	98-05-01	23.37
Boiling Point, degrees Celsius, range, upper bound	NCEC	98-05-01	23.41
Boiling Point, degrees Fahrenheit	ChemTox	97-09-01	23.2
Breathing apparatus required	LoLi	98-07-01	Yes
Degrees, Doctoral, Ph.D., Indexed by author	LoLi	94-01-01	G. Plimpton
Melting Point, degrees Celsius	ISIS	97-09-01	13.2

By describing them in the above fashion, the values are grouped together and will be displayed next to each other, so that the user can see all of the available values clearly and unambiguously.

If the Field Descriptions were less orderly, the data could be spread out across a very large table, and the user would not necessarily know that related data exists under a different heading. Again, as an example, look at the following:

Field Description	Authority	Last Updated	Value
Boiling Point, degrees Celsius	ISIS	98-01-01	23.25
Breathing apparatus required	LoLi	98-07-01	Yes
Degrees Celsius, Boiling Point	LoLi	98-07-01	23.3
Degrees Celsius, Boiling Point	LoLi	97-10-01	23.2
Degrees, Doctoral, Ph.D., Indexed by author	LoLi	94-01-01	G. Plimpton
Degrees Fahrenheit, Boiling Point	ChemTox	97-09-01	23.2
Melting Point, degrees Celsius	ISIS	97-09-01	13.2
Range, lower bound, degrees Celsius, Boiling Point	NCEC	98-05-01	23.37
Range, upper bound, degrees Celsius, Boiling Point	NCEC	98-05-01	23.41

In the above example, because the Field Descriptions are not assigned in an ordered fashion, the related data is spread out all across the data set and is much more difficult to comprehend, let alone permit the user to see if related data is available.

Queries

Apart from the two methods of accessing the data discussed above, there are so far two types of queries for substance-related data that can be envisaged at this time:

- Encyclopedic (i.e. "*Provide the toxicological properties of specified substances relating to mutagenicity in rats.*").
- Analytical (i.e. "*Which furan-based chemicals on the Toxic Release Inventory list are vented into the atmosphere in the Southeastern United States in quantities greater than 10 tons/year.*")

Document data (non substance-related) can be queried using a classical full-text search and retrieval method. (*Find all documents relating to maritime transportation of petrochemicals within the jurisdiction of navigable Canadian waters.*)

Certain types of documents, for example MSDSs, merit being queried by both methods. (*Select all MSDSs for substances that are not on the Canadian Domestic Substances List that contain the phrase* 'flush eyes with water' *in the First-Aid section.*)

A mock-up of a proposed screen layout for the DataCHEST Query Wizard can be found below.

The Query Wizard will be eventually implemented as a Java Applet that will run within a compatible browser, such as Netscape or MsIE. The Wizard consists of a main window with 3 principal sub-windows.

The main window contains various menu bars and toolbars, consisting of buttons and controls to assist in navigating within the sub-windows.

The first and second sub-windows, respectively entitled Shopping Mall and Data Store, are intended to function much like MsWindows Explorer, mainly so that most users will already be familiar with the principles of its operation. The function of these two windows will be to assist the user in building the desired query segments.

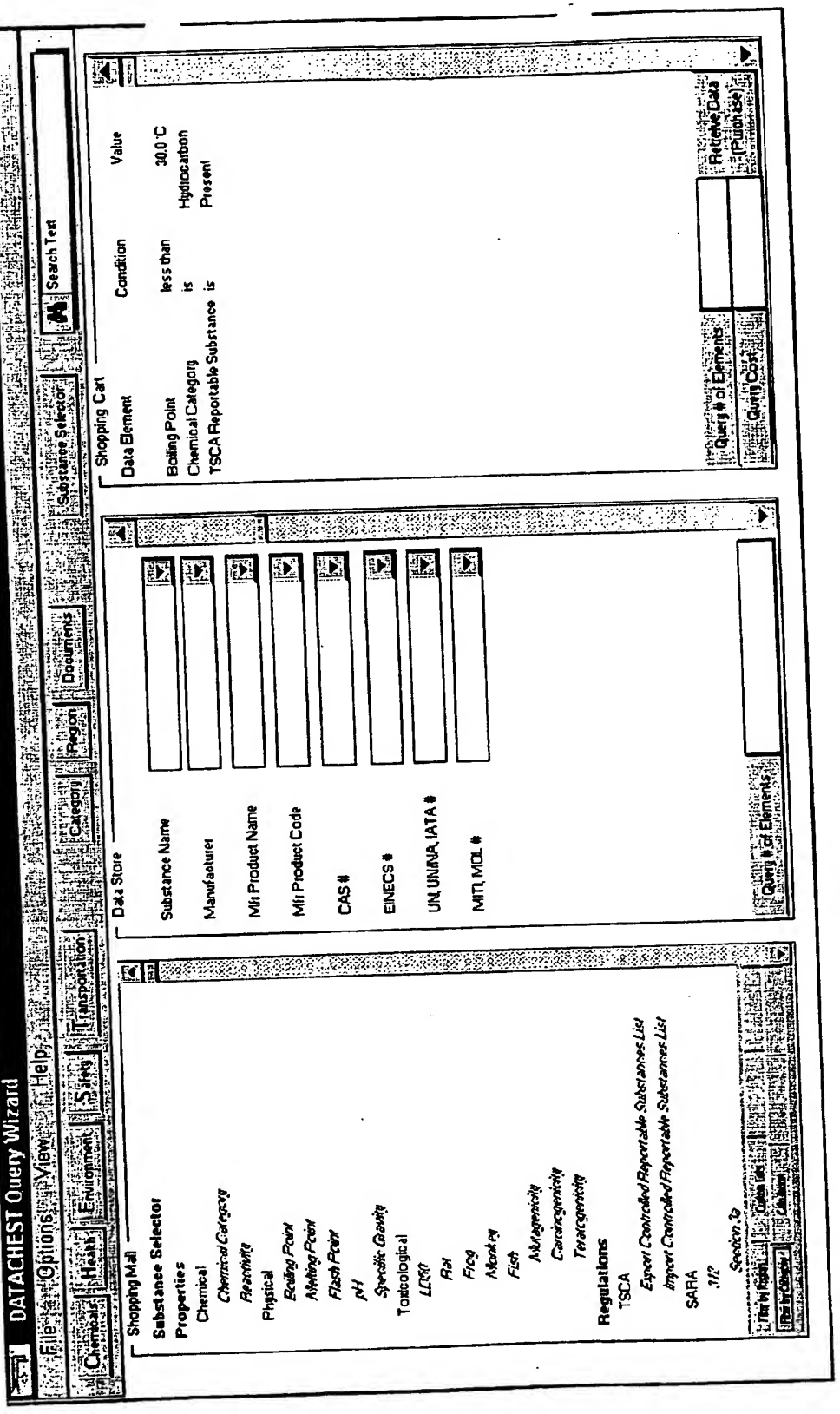
The third sub-window, entitled Shopping Cart, will hold the query segments as each one is completed and will enable the user to process the final query.

As with MsWindows Explorer, the Shopping Mall sub-window is organized as a tree structure containing the field groups and descriptions contained in the DataCHEST database. Unlike Explorer however, the different levels of the tree are variable, and can be selected by the tab control at the bottom of the sub-window. The user can thus choose the order in which to drill down through the levels of the tree. The buttons will also pre-select an ordering of the tree levels. The search control on the toolbar shown at the top of the main window will retrieve all field descriptions and groups whose contents match that of the search window, according to the currently selected ordering of the tree.

The Data Store sub-window will change according to the type of field that is currently selected in the Shopping Mall sub-window. Obviously, there will be a number of pre-defined window types, with parameters for naming the various selection controls. In the example below, a configuration that might be suitable for selecting a substance has been shown. A numeric data value may call up a configuration that allows the user to specify a range within which the value must fall. A text data value may call up a configuration that allows the user to specify a string that should contained within, begin or end the data value. Other data value types may have other configurations.

In some instances it will be useful for the user to be able to ask for the number of data elements which match the constraints imposed as a query segment is built. A button will be provided for this purpose within the Data Store sub-window. Under certain circumstances a fee may be imposed for retrieving this information.

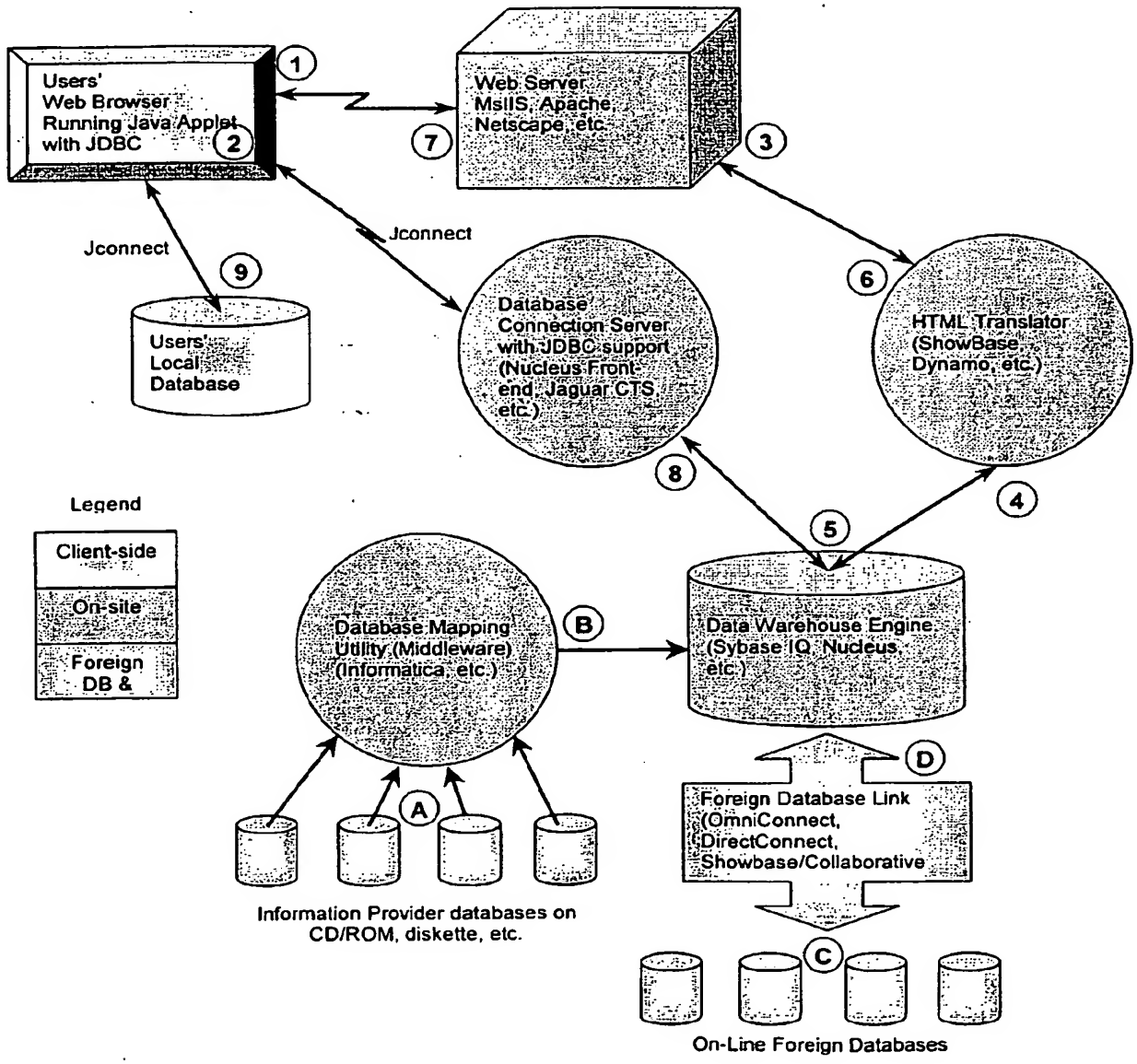
The Shopping Cart sub-window will hold the contents of each sub-query. As with the Data Store sub-window, there will be a button control that will retrieve the hit count, although in this instance, it is more likely that a fee will be imposed for retrieving this information. Another button at the bottom of the sub-window will calculate the cost of retrieving the query (this will not be chargeable). A final set of buttons will retrieve the data elements associated with the query. The first button will retrieve the query in a visual format as a Web Page. The other button will retrieve the query as a result data set that can be stored locally on the user's workstation, as long as the DataCHEST client software has been purchased and installed.



Functional Organization

The following diagram describes our preliminary view of how the various parts of the database will be organized. Obviously, until the choice of tools has been confirmed, the functional organization cannot be frozen.

Figure 3: Functional Organization



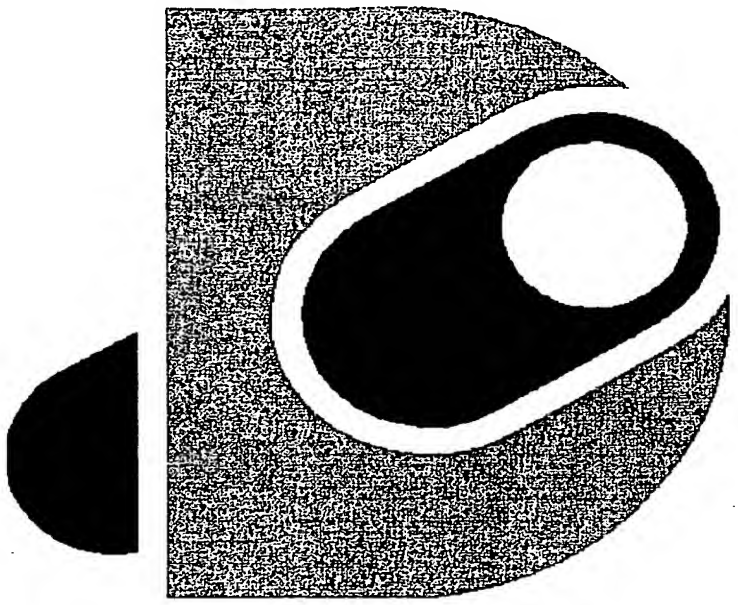
Clients will access the DataCHEST server from their Internet Web Browser **(1)**. From there, they will select which type of query they want to perform, either an interactive one displaying results on the screen, or a query which will return a result data set as an SQL table. They can then design their query interactively using a Wizard, or else choose to supply their own SQL statements and expressions.

In each case, a Java or ActiveX applet will be downloaded to their browser **(2)** to execute the query wizard or receive the SQL statements. Interactive queries will be passed from the Web Server **(3)** to an HTML translator **(4)** that will transmit the query to the Data Warehouse **(5)**. The Data Warehouse will process the query and retreive the result set, passing it back to the HTML translator where it will be formatted into an HTML page **(6)** that will be passed back to the Web Server and sent **(7)** to the Web Browser.

SQL table queries will connect directly to a Database connection server **(8)** that will maintain and control the concurrent sessions, threads and access security, passing the query to the Data Warehouse **(5)** and returning the result set to the Browser Applet **(2)**. The Browser Applet will in turn store the result set in a local database **(9)**.

Periodically, as the Information Providers update their data, their data sets **(A)** will be imported into the master database in the Data Warehouse **(5)**. A suitable tool **(B)** will be used to map the source data onto the master target, so that the process can be repeated each time, making only those changes to the established mappings that are required to reflect updates to the structure of the source databases from the Information Providers.

The possibility of collaborating with remote databases **(C)**, maintained by the Information Providers at their location should also be available, although it is doubtful if this level of complexity can be achieved at the outset. To do this would require a Foreign Database Linkage module **(D)**.



DataCHEST.com

DataCHEST Data Repository -- Data Model

Table of Contents

Abstract

This document describes the data model employed by DataCHEST for its Data Repository

Design Goals

The DataCHEST Data Repository is a data store for data originating from a wide variety of sources. As such, it must provide a simple and cohesive structure in wnich to catalogue and index the transformed source data.

The DataCHEST Data Repository is intended to be a reference source rather than a transactional database. As such, the data must be easily retrievable by a wide variety of criteria, instead of being organised according to a predetermined logic.

Data Types

The integration of each new database into the Data Repository has the potential to justify the need for the creation of a new data type in order to manage the data to be stored. Hopefully, the process of transforming and importing data from the initial set of databases that have been evaluated will result in the definition of sufficient variety of data types to handle the transformation of the majority of databases that will be encountered during the first year of deployment.

Each of the data types contemplated so far are listed and discussed below.

Enfielded, relational data elements

The primary characteristic of this type of data that will be imported into the DataCHEST Data Repository is that it is substance-related. The data elements will always relate to a particular substance, for example,

- the boiling point of Ethanol, or
- the fact that Freon belongs to the family of chlorinated hydrocarbons, or
- that Freon is listed as an environmental hazard for atmospheric release under the US Federal Clean Air Act.

Enfielded, relational data is by far the most complicated type of data that will have to be transformed into a unified structure. The primary reason for this is that the source data already exists within its own structure, and is already linked to other elements by the relationships within that structure.

The key to integrating data from these various dissimilar structures is to create a database schema in which the data element definitions are themselves data. This strategy ensures that the data element definitions are thus extensible and flexible enough to accommodate each new type of data element and each new structure as new data sources are imported into the Data Repository.

Data in this category will fall into four major types: Boolean, Numeric, Text Strings and Large Objects.

Boolean data

Boolean data represents an either-or binary value. In the context of the DataCHEST Data Repository, the data represented will have meanings like True/False, Yes/No, and Presence on or Absence from a list.

Numeric data

Numeric data will be represented as a floating-point value. There does not seem to be enough differentiation to warrant the segregation of integer values in a separate table.

Text strings

Text strings will be limited to a length of 255 characters in order to remain compatible with all of the common ODBC-compatible database engines available on the market today. Textual database elements that exceed this size will be stored as Large Objects, even though their indexing process might require some time-consuming conversions or the use of a full-text search engine.

Text Strings will also be used to store hyperlink data elements, or pointers to objects stored elsewhere.

Large objects

Large objects will be used to store data that, although it is relational, cannot be stored in the three conventional data types discussed above. Some of the data types that will fall into this category are discussed below.

It has not yet been decided whether to store all large objects in the Data Repository within the Database Engine itself, or else as files in a file system. The indexing and linking processes should be able to support either option.

Object data elements

It will be difficult, in certain instances, to convert the data into common forms that will permit direct comparison of values between providers. Some providers store numeric data in numeric form, while other have chosen to store numeric data as text in order to handle exceptions or unusual values. DataCHEST must endeavour, in addition to publishing the data in the providers' original format, to do its utmost to provide uniform data formats wherever possible.

Thus, for simple numeric attributes, such as boiling points, there will not be one simple numeric element type that can completely describe the possible values. Rather, there will have to be a series of elements, organised into an object structure, which completely describe the attribute. A possible example of an object structure for describing boiling points is provided below.

1. Numeric value for the temperature
2. Unit of measure Token (which points to a multi-lingual list of values, e.g. °C, °F, °K, etc.)
   (Or else convert all values to a common unit of measure and re-convert them to a specific unit of measure upon retrieval)
3. State Change Token (which points to a multi-lingual list of values, e.g. Boils, Decomposes, Explodes, etc.)
4. Numeric value for the pressure at which the measurement was taken

5. Unit of measure Token (points to a multi-lingual list of values, e.g. mm/Hg, psi, KG/cm^2, atmospheres, etc.)

While this may seem sufficient to cover the various easily imaginable cases, there will inevitably be exceptions to even these broad specifications. For that purpose, we have included with every data element, the option of including a remark or comment in which the circumstances of the particular exception can be explained.

Unstructured textual data elements

For the most part, unstructured textual data elements are not substance-related, and consist of texts of law, regulations, scientific and medical research studies, news reports and other documents. These data elements will be indexed using a full-text indexing search engine.

On the other hand, a substantial number of textual data elements, such as export certificates, patents, and MSDSs, are substance-related. These documents can be indexed in two ways: as full-text and as hyperlink values in a relational database.

Semi-structured textual data elements

MSDSs are also representative of semi-structured data elements in that they can be divided into meaningful zones or areas of text that have a certain significance. For instance, in some MSDS formats, Section 2 might contain information about the ingredients of mixtures, as well as physical and chemical properties, while Section 4 might contain First Aid Procedures. By pre-processing these documents prior to storage in the repository, DataCHEST can include zone markers that will permit users to search within a specific zone inside each MSDS. This capability can add valuable information to the context of a query.

Spatial data elements

Although this type of information may not be useful in the initial releases of the DataCHEST Data Repository, later releases that include this data type will provide valuable functionality to the users. Using spatial information, users will not only be able to select regional data elements by clicking on maps, but they will also be able to do things like build diagrams of 2D and 3D representations of molecules or molecule fragments and search for matches across the substance database. Statistical or transactional data could also be summarised by region and displayed on a map using a coloured intensity-scale.

Image data elements

Various types of elements data fall into this category, such as pictograms, scanned documents, satellite photographs and X-ray images. For the most part, these data elements will have to be indexed using keywords, added either manually, or derived from the file name and/or source directory. One notable exception to this rule is the case of scanned documents that could be processed through an OCR (Optical Character Recognition) engine and full-text indexed.

Other data element types

Despite the relatively exhaustive list of standard data element types described above, there are still other data types that will have to be handled in the DataCHEST Data Repository. Audio and video clips, Multimedia objects, statistical and transactional data will all find their way into the DataCHEST melting pot following the initial release of the application.

Multi-lingual data

The textual data elements and documents contained in the DataCHEST Data Repository originate from all over the world, and are written in different languages. Both the Data Repository and the Query Wizard must be able to search across these data elements, regardless of language, and must also be able to manage and display the data.

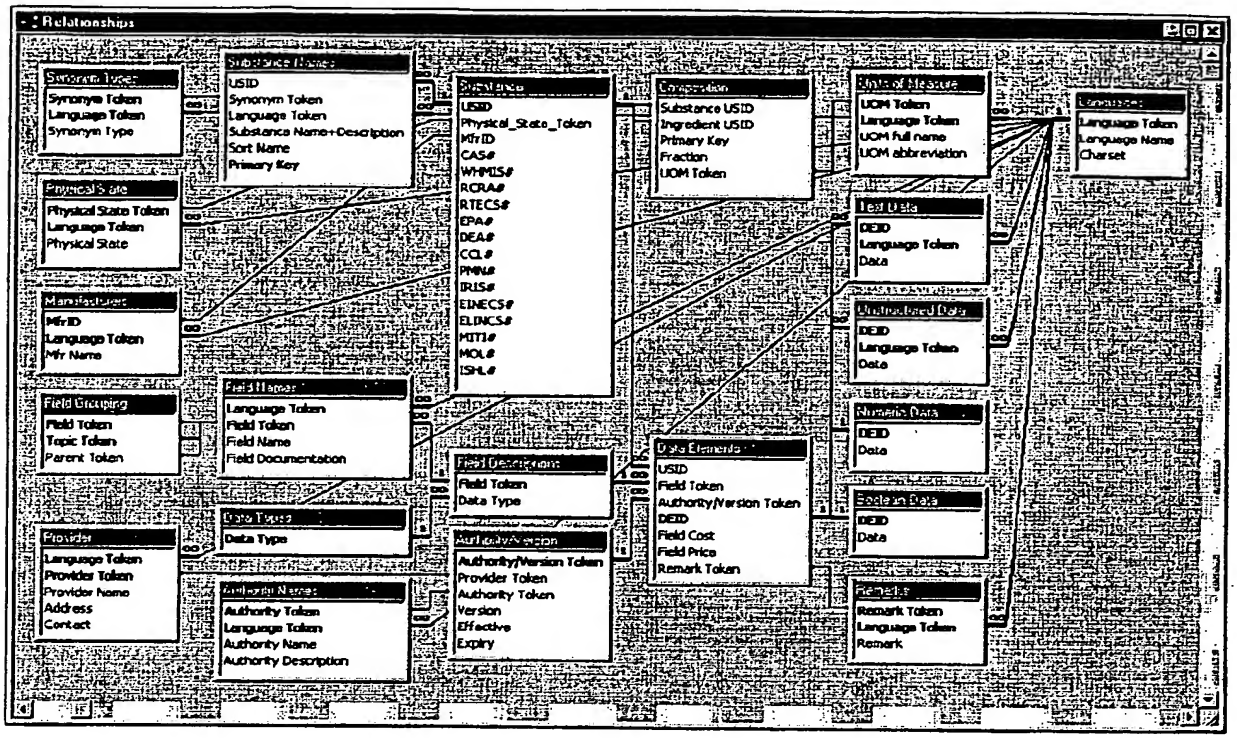
The schema of the Data Repository must thus be structured so that all of the text elements can be identified by language, and furthermore, so that they can be translated and stored in more than one language.

Schema

Schema Diagrams

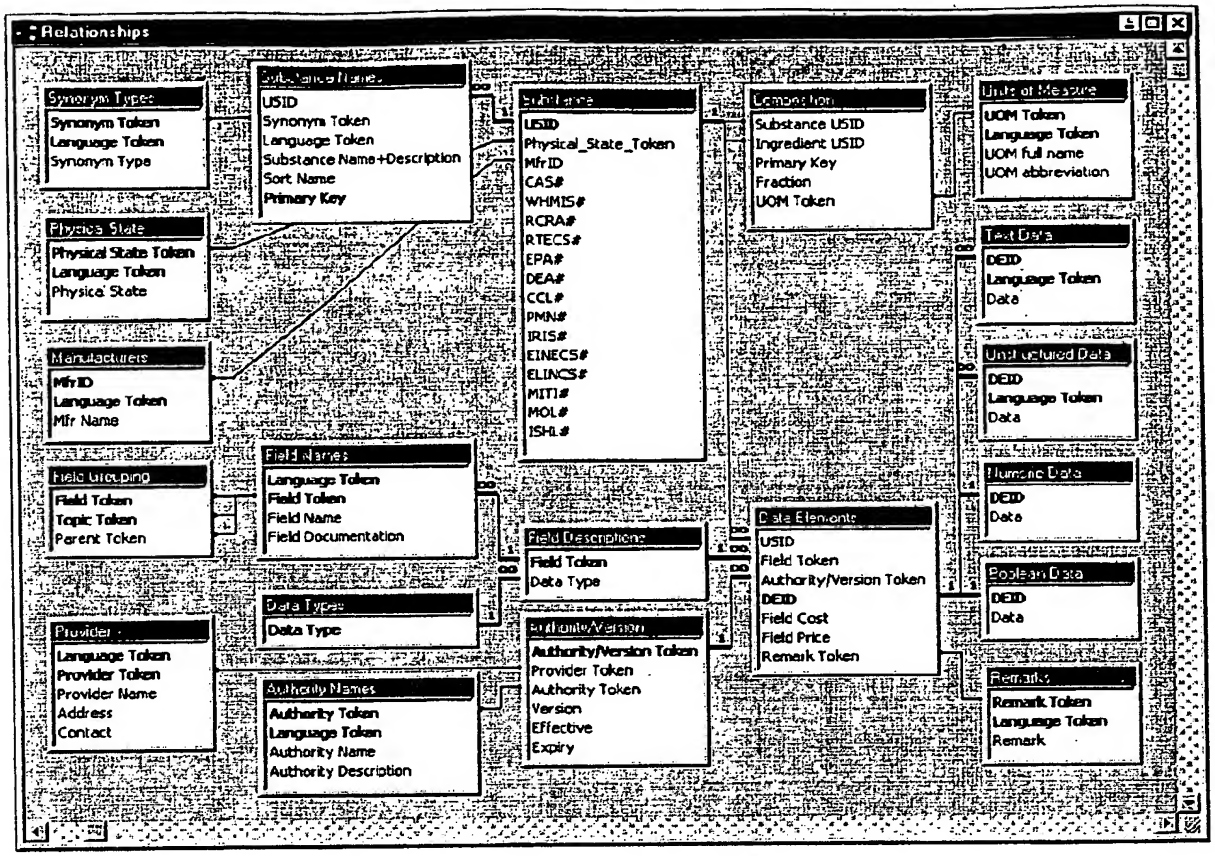
The Schema of the Data Repository is diagrammed in the two figures below. The first figure includes the schema with the language relationships displayed, and demonstrates the level of complexity that is added by the multi-lingual nature of the data elements.

Figure 1 – DataCHEST Data Repository Schema, with language relationships displayed



The second figure, below, diagrams the same schema without displaying the language relationships, making the structure much more easily discernible.

Figure 2 – DataCHEST Data Repository Schema, with language relationships hidden



Schema Description

The schema is centered on the **Data Elements** table, which uniquely identifies, but does not contain, each data element. Data elements are identified by a Data Element Identifier (**DEID**) and are stored in separate tables according to their data type, and where applicable, the language in which they are written. This enables the repository to manage data that exists in or has been translated into multiple languages, for example, First Aid Procedures, MSDS's or TREMCards.

Each data element is uniquely identified by three criteria. A Unique Substance Identifier (**USID**) relates the data element to a specific substance, a **Field Token** relates to a specific Field Description which defines the data element, and an **Authority/Version Token** specifies the source of the data element.

Data elements can also reference individual **Remarks** contained in their own table. These are comments that would apply to the individual data element, noting an exception to the general rule for the definition of the data element, for example, that a boiling point for carbon dioxide cannot be obtained at standard pressures because the substance evaporates directly from its solid state.

The **Substance** table uniquely identifies each substance in the Data Repository. Because a substance can have different properties depending upon its **Physical State** or who the **Manufacturer** is, there could frequently be more than one record in the database for a given chemical compound. To facilitate searching

and retrieval, attributes that uniquely identify a substance will be contained in the **Substance** table (the ones shown are for example only, and more will be added).

The **Substance Names** table provides an important tool to facilitate searching for substances. Once again, due to multi-lingual considerations, the names are contained in a separate table. It is extremely common for a substance to have multiple names, and this characteristic is handled by referring each name to an entry in the **Synonym Types** table.

Substance Names are often vague, referring at times to a unique substance and at other times, referring to a class of substances. By allowing duplicate **Substance Names**, the structure will provide for this condition. If the duplication of names proves to be unworkable, the relationship between **Substance** and **Substance Names** can be modified to implement a many-to-many relationship.

A **Composition** table provides a means by which it is possible to represent mixtures by listing each ingredient (which itself must exist in the Substance Table) together with its concentration. Note that the data will simply be imported from the Information Provider. No attempt will be made to ensure that the total ingredient concentration will add up to 100%. Concentrations can be specified in any desired unit of measure, as referenced by the **Units of Measure** Table.

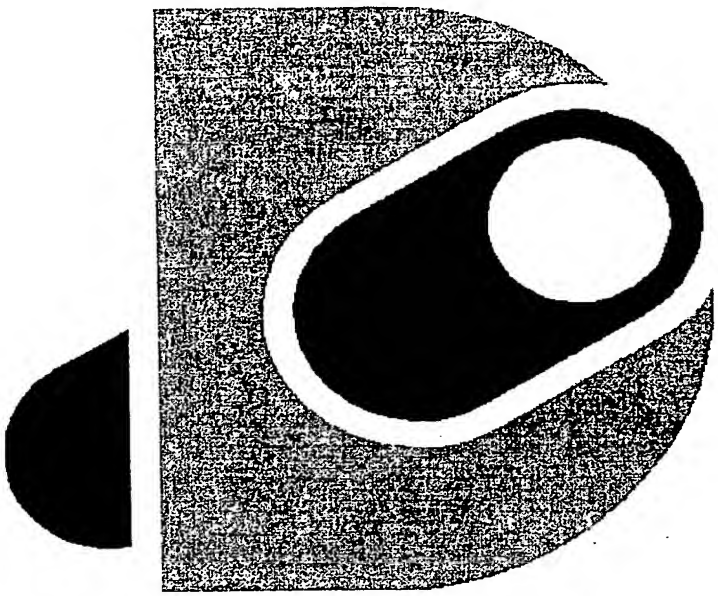
A **Field Descriptions** table identifies the various fields that have been defined for substance-related data. To handle multi-lingual **Field Names**, these values are stored in a separate table. Each **Field Description** is associated with a unique **Data Type**, which is related to the data table in which the data is stored. For clarity, the Data Model is shown with four major **Data Types (Boolean, Numeric, Text**, and **Unstructured)**, but additional types are also contemplated. Examples of these would be numeric ranges, hyperlinks, and other types of data.

The **Field Grouping** table organises the **Field Descriptions** into the multi-level tree-structure for easier searching and selection.

An **Authority/Version Table** identifies the reference source **(Authority Names)** of the data as well as the Information **Provider** that supplied the data, together with the version and date of last update of their source database.

A **Languages** Table lists the various supported languages and is keyed to those tables that are able to hold multilingual data.

Title : DataCHEST Applicati. Design Specifications
Created : 2000-02-11 15:02 By : Alan S. Lawee
Last modified : 2000-02-11 15:09 By : Alan Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\DATACHEST APPL CATION DESIGN SPECS CLEAN.DOC



DataCHEST.com

DataCHEST Application Design Specifications

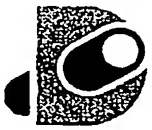
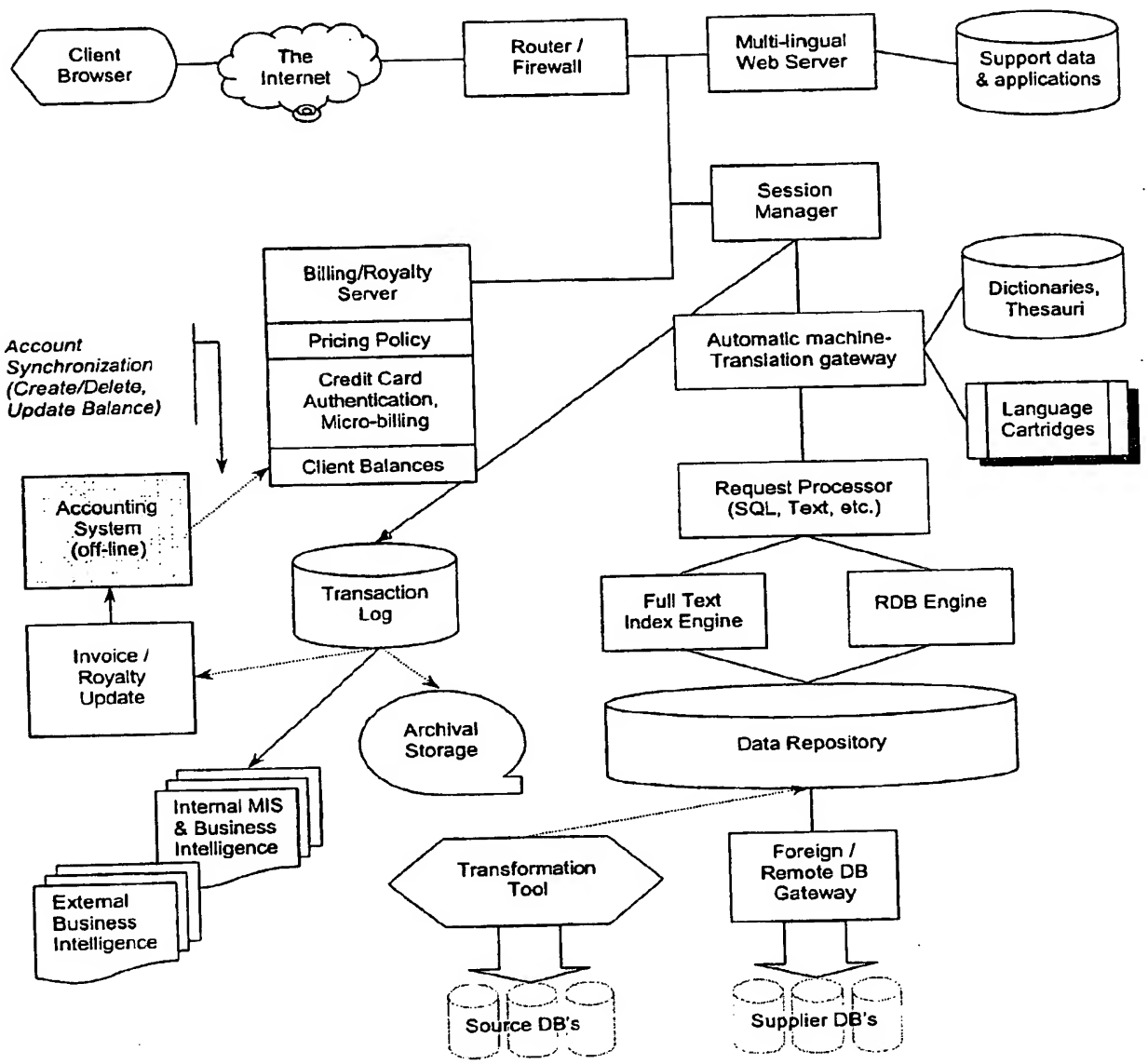


Table of Contents

Abstract

This document sets out the design specifications for the development and implementation of a Knowledge Management and Distribution System (KMDS) to facilitate the importing of data to the DataCHEST data repository at the back end and the retrieval of the data from the repository at the front end. The system will also manage the client billing, the supplier royalty payments and the transaction processing and analysis.



OL_DATA_ELEMENTS Table Creation Script

```
CREATE TABLE OL_DATA_ELEMENTS
 (ELEM_ID NUMBER(10) NOT NULL
 .SFID NUMBER(7) NOT NULL
 .VERSION_TOKEN NUMBER NOT NULL
 .FIELD_TOKEN NUMBER(7) NOT NULL
 .DEID NUMBER(12) NOT NULL
 .PARENT_DEID NUMBER(12) NOT NULL
 .FIELD_COST NUMBER(8,2) NOT NULL
 .FIELD_PRICE NUMBER(8,2) NOT NULL
 .EFFECTIVE DATE
 .EXPIRY DATE
 .INTEGRATED DATE NOT NULL
 .RMK_TOKEN NUMBER(12)
 .MFRID NUMBER(12) NOT NULL
 .PHST_TOKEN NUMBER(12) NOT NULL
 .USID NUMBER(7) NOT NULL
 .MFR_NAME VARCHAR2(255) NOT NULL
 .PHYSICAL_STATE VARCHAR2(255) NOT NULL
 .SU$VERSION_TOKEN NUMBER NOT NULL
 .CAS VARCHAR2(20) NOT NULL
 .SRCDB_ID VARCHAR2(255) NOT NULL
 .SDBID_DESCR VARCHAR2(4000)
 .DC_DATE DATE NOT NULL
 .DTATYP_TOKEN NUMBER(3) NOT NULL
 .PV_TOKEN NUMBER(12) NOT NULL
 .AV_TOKEN NUMBER(12) NOT NULL
 .AUTH_TOKEN NUMBER(12) NOT NULL
 .AV$DC_VERSION VARCHAR2(100) NOT NULL
 .AV$EFFECTIVE DATE
 .AV$EXPIRY DATE
 .AV$CREATED DATE NOT NULL
 .AUTH_NAME VARCHAR2(255) NOT NULL
 .PV$DC_VERSION VARCHAR2(100) NOT NULL
 .PROV_TOKEN NUMBER(12) NOT NULL
 .PV$EFFECTIVE DATE
 .PV$EXPIRY DATE
 .PV$CREATED DATE NOT NULL
 .PROV_NAME VARCHAR2(255) NOT NULL
 .PROV_PRODUCT_NAME VARCHAR2(255) NOT NULL
 .LTID NUMBER(7)
 .FD$ICONS_TOKEN NUMBER(5)
 .FD$DTATYP_TOKEN NUMBER(3) NOT NULL
 )
```

Primary Key

```
ALTER TABLE OL_DATA_ELEMENTS
 ADD CONSTRAINT OL_DATA_ELEMENTS_PK PRIMARY KEY
  (ELEM_ID)
```

Note : The $ sign in a field name is use to separate the table ALIAS from the field name used in the creation SQL statement for a field name that is common in two tables (i.e. PROVIDER_VERSION « PV » EXPIRY date vs AUTHORITY_VERSION « AN » EXPIRY date)

Data loading logic

The OL (On Line) Data Element table is populated from the MAIN database using the following tables (*Note : Not all fields of a given table may be used*):

Table Names	Field Names	Field Descriptors	
AUTHORITY_NAME	AUTH_TOKEN	NOT NULL	NUMBER(12)
	AUTH_NAME	NOT NULL	VARCHAR2(255)
AUTH_VERSIONS	AV_TOKEN	NOT NULL	NUMBER(12)
	AUTH_TOKEN	NOT NULL	NUMBER(12)
	DC_VERSION	NOT NULL	VARCHAR2(100)
	EFFECTIVE	NOT NULL	DATE
	EXPIRY		DATE
	CREATED	NOT NULL	DATE
DATA_ELEMENTS	ELEM_ID	NOT NULL	NUMBER(12)
	SFID	NOT NULL	NUMBER(7)
	VERSION_TOKEN	NOT NULL	NUMBER
	FIELD_TOKEN	NOT NULL	NUMBER(7)
	DEID	NOT NULL	NUMBER(12)
	PARENT_DEID	NOT NULL	NUMBER(12)
	FIELD_COST	NOT NULL	NUMBER(8,2)
	FIELD_PRICE	NOT NULL	NUMBER(8,2)
	EFFECTIVE		DATE
	EXPIRY		DATE
	INTEGRATED	NOT NULL	DATE
	RMK_TOKEN		NUMBER(12)
ELEMS	DEID	NOT NULL	NUMBER(12)
	DTATYP_TOKEN	NOT NULL	NUMBER(3)
FIELD_DESCRIPTIONS	FIELD_TOKEN	NOT NULL	NUMBER(7)
	LTID	NOT NULL	NUMBER(7)
	ICONS_TOKEN		NUMBER(5)
	DTATYP_TOKEN	NOT NULL	NUMBER(3)
INFO_VERSIONS	VERSION_TOKEN	NOT NULL	NUMBER
	PV_TOKEN	NOT NULL	NUMBER(12)
	AV_TOKEN	NOT NULL	NUMBER(12)
MANUFACTURERS	MFRID	NOT NULL	NUMBER(12)
	MFR_NAME	NOT NULL	VARCHAR2(255)
PHYSICAL_STATES	PHST_TOKEN	NOT NULL	NUMBER(12)
	PHYSICAL_STATE	NOT NULL	VARCHAR2(255)
PROVIDERS	PROV_TOKEN	NOT NULL	NUMBER(12)
	PROV_NAME	NOT NULL	VARCHAR2(255)
	ADDRESS		VARCHAR2(255)
	PROV_PRODUCT_NAME		VARCHAR2(255)
	CONTACT		VARCHAR2(255)

PROVIDER_VERSIONS	PV_TOKEN	NOT NULL	NUMBER(12)
	DC_VERSION	NOT NULL	VARCHAR2(100)
	PROV_TOKEN	NOT NULL	NUMBER(12)
	EFFECTIVE		DATE
	EXPIRY		DATE
	CREATED	NOT NULL	DATE
SUBSTANCES	USID	NOT NULL	NUMBER(7)
	VERSION_TOKEN	NOT NULL	NUMBER
	CAS	NOT NULL	VARCHAR2(20)
	SRCDB_ID	NOT NULL	VARCHAR2(255)
	SDBID_DESCR		VARCHAR2(4000)
	DC_DATE	NOT NULL	DATE
SUBST_FORMS	SFID	NOT NULL	NUMBER(7)
	MFRID	NOT NULL	NUMBER(12)
	PHST_TOKEN	NOT NULL	NUMBER(12)
	USID	NOT NULL	NUMBER(7)
SUBST_NAMES	LANG_TOKEN	NOT NULL	NUMBER(3)
	NAME_TOKEN	NOT NULL	NUMBER(9)
	SORT_NAME	NOT NULL	VARCHAR2(255)
	NAME_DESCRIPTION		VARCHAR2(4000)
	DATE_CREATED	NOT NULL	DATE
	VERSION_TOKEN	NOT NULL	NUMBER
	SYN_TOKEN	NOT NULL	NUMBER(7)

Creation SQL statement

In order to load the data, the following SQL statement must be executed using a cursor and a commit point (In order to avoid Rollback Segments to crash...)

```
SELECT DE.ELEM_ID,
       DE.SFID,
       DE.VERSION_TOKEN,
       DE.FIELD_TOKEN,
       DE.DEID,
       DE.PARENT_DEID,
       DE.FIELD_COST,
       DE.FIELD_PRICE,
       DE.EFFECTIVE,
       DE.EXPIRY,
       DE.INTEGRATED,
       DE.RMK_TOKEN,
       SF.MFRID,
       SF.PHST_TOKEN,
       SF.USID,
       MA.MFR_NAME,
       PS.PHYSICAL_STATE,
       SU.VERSION_TOKEN SU$VERSION_TOKEN,
       SU.CAS,
       SU.SRCDB_ID,
       SU.SDBID_DESCR,
       SU.DC_DATE,
       EL.DTATYP_TOKEN,
       IV.PV_TOKEN,
       IV.AV_TOKEN,
       AV.AUTH_TOKEN,
       AV.DC_VERSION AV$DC_VERSION,
       AV.EFFECTIVE AV$EFFECTIVE,
       AV.EXPIRY AV$EXPIRY,
       AV.CREATED AV$CREATED,
       AN.AUTH_NAME,
       PV.DC_VERSION PV$DC_VERSION,
       PV.PROV_TOKEN,
       PV.EFFECTIVE PV$EFFECTIVE,
       PV.EXPIRY PV$EXPIRY,
       PV.CREATED PV$CREATED,
       PR.PROV_NAME,
       PR.PROV_PRODUCT_NAME,
       FD.LTID,
       FD.ICONS_TOKEN FD$ICONS_TOKEN,
       FD.DTATYP_TOKEN FD$DTATYP_TOKEN
  FROM SUBST_FORMS SF,
       MANUFACTURERS MA,
       PHYSICAL_STATES PS,
       SUBSTANCES SU,
       ELEMS EL,
       INFO_VERSIONS IV,
       AUTH_VERSIONS AV,
       AUTHORITY_NAME AN,
       PROVIDER_VERSIONS PV,
       PROVIDERS PR,
       FIELD_DESCRIPTIONS FD,
       DATA_ELEMENTS DE
```

```
WHERE DE.SFID = SF.SFID
  AND SF.MFRID = MA.MFRID
  AND SF.PHST_TOKEN = PS.PHST_TOKEN
  AND SF.USID = SU.USID
  AND DE.DEID = EL.DEID
  AND DE.VERSION_TOKEN = IV.VERSION_TOKEN
  AND IV.AV_TOKEN = AV.AV_TOKEN
  AND AV.AUTH_TOKEN = AN.AUTH_TOKEN
  AND IV.PV_TOKEN = PV.PV_TOKEN
  AND DE.FIELD_TOKEN = FD.FIELD_TOKEN
  AND PV.PROV_TOKEN = PR.PROV_TOKEN;
```

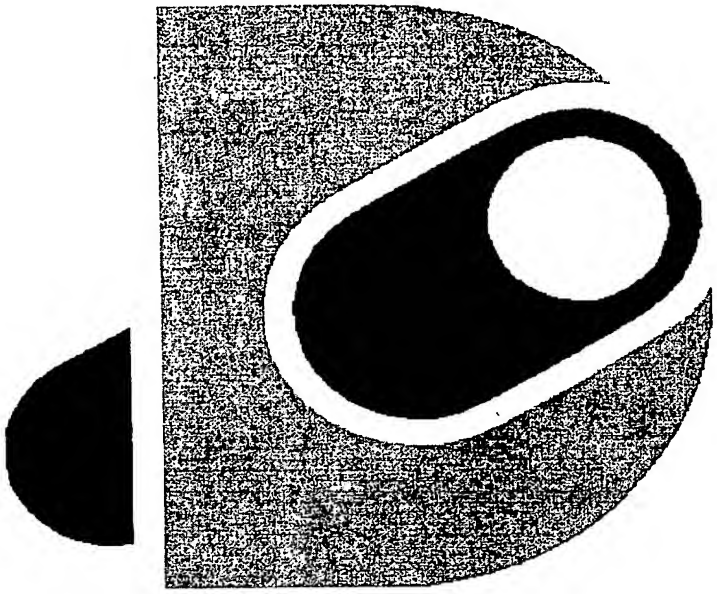
Quality Check

To ensure that the On Line Data Element table has been populated correctly, the following rules or guidelines must be verified :

1. The number of records in the DATA_ELEMENTS table must be the same as the target OL_DATA_ELEMENTS table.
2. If the number of record created is less (ref. rule #1) a Foreign Key Integrity Constraint has been violated when generating the data (i.e. a foreign exist in one table but not in the referenced table)
3. If the number of record equals 0, one (or more) of the table is empty (Same violation as rule #3 but easier to identify).
4. If the number of record in the DATA_ELEMENTS is less than the newly generated OL_DATA_ELEMENTS tables, it means that a Primary Key constraint as been violated (or disabled).

Title : Information Manager And Distribution System (IMADS) Design Specifica
Created : 2000-02-11 15:02 By : Alan S. Lawee
Last modified : 2000-02-11 15:16 By : Alan S. Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\IMDS DESIGN SPECS CLEAN.DOC

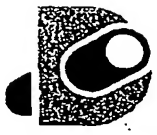
Information Management And Distribution System



DataCHEST.com

(IMADS) Design Specifications

Table of Contents



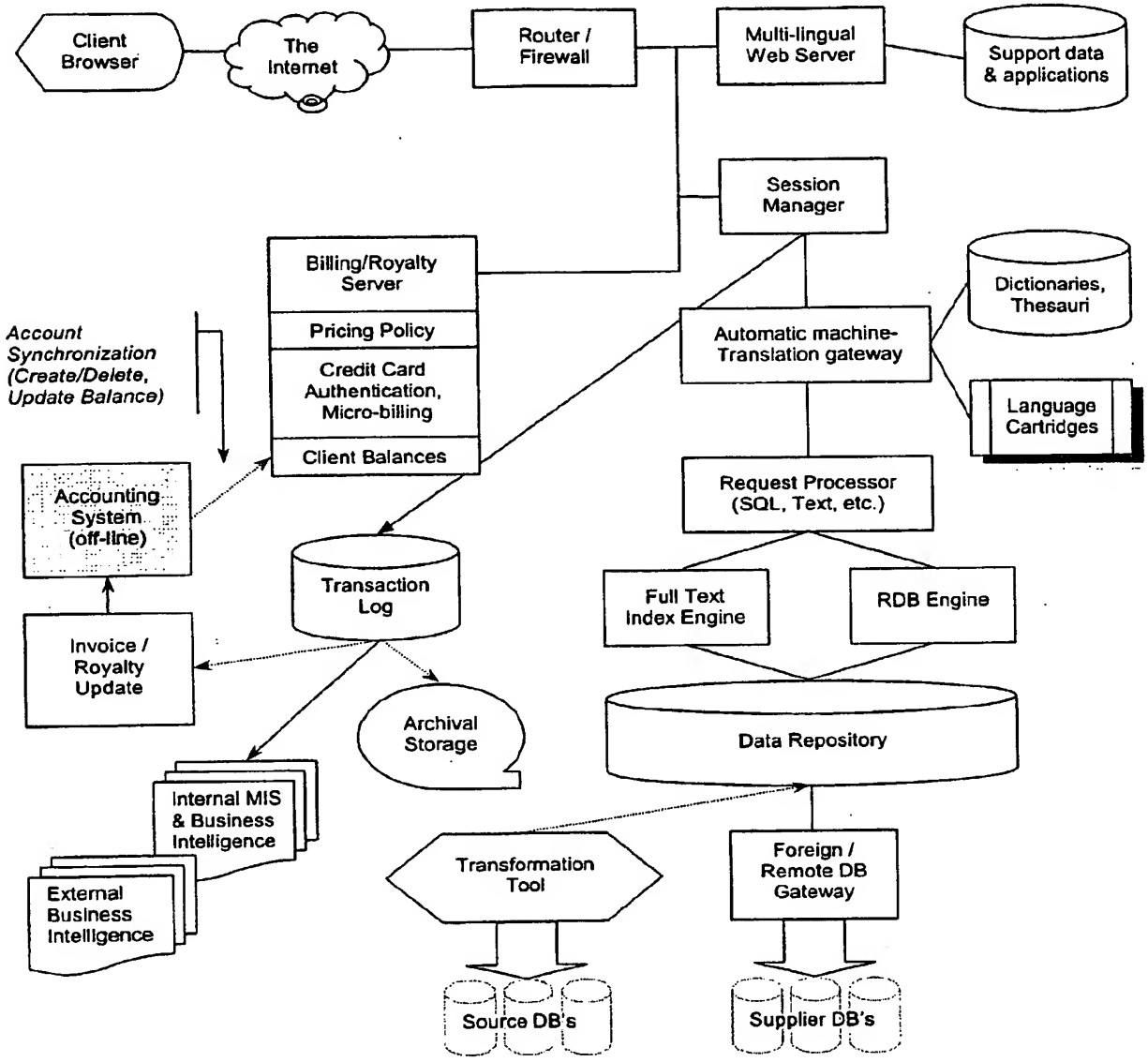
Abstract

This document sets out the design specifications for the development and implementation of a Knowledge Management and Distribution System (KMDS) to facilitate the importing of data to the DataCHEST data repository at the back end and the retrieval of the data from the repository at the front end. The system will also manage the client billing, the supplier royalty payments and the transaction processing and analysis.

Minimum set of capabilities required for the application

The minimum set of operational capabilities required for the DataCHEST IMADS is listed below.

1. The IMADS must be able to store, index, and retrieve all types of data formats. These can range from simple Boolean (True/False), Numeric (Integer or Floating point), or Text values, to a complex array of values (Objects), bit-mapped images, formatted documents or other data types.
2. The IMADS must be able to process multiple languages and character sets. DataCHEST source databases will originate from countries throughout the world, and must be accessible to any client who requires the data. DataCHEST clients will understand many different languages and must be able to use the IMADS in their native language if their business case warrants translation of the user interface.
3. The size of the database and the number of data elements it will contain will be extremely large.
4. The IMADS must be capable of being mirrored to multiple remote sites both in toto and in subsets.
5. The IMADS client must have an open architecture and be able to be executed from all of the popular platforms, including at a minimum, MsWindows (3.1, '95, '98, 'NT), MacOS (System 7 and higher), and Unix (with a graphics terminal). Support for other platforms, for example IBM AS/400, would be desirable. Of course it might be impractical to assume that the user interface would function properly on a character-based terminal.
6. The IMADS server must also have an open architecture. Its modular nature provides the flexibility for certain application segments to run on different platforms. While key components, like the transaction server or the Web server might be limited to running on Unix or Windows NT Server, the Data Repository should be capable of being run in a variety of environments
7. The data transformation tool must be able to import data from all popular database platforms, flat file formats, and popular spreadsheet and word-processor formats.
8. The Query building process must be intuitive, convenient and simple to use.
9. Query retrieval must offer comfortable response times. The user interface should be able to analyse user queries and warn the user if his request is unusually complex or large, or if it will take a long time to execute. Ideally, such requests should be able to be deferred and run remotely, with large result data sets being returned off-line, either by wire or by hard media, such as CD/ROM or tape.



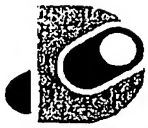
Definition

The following is how a definition should appear in documents.

Template
A special kind of document that provides basic tools for shaping a final document.

Defined terms should be Indexed.

The definition should appear in a glossary at the end of a document.



Glossary

To fill in the glossary, do the following:

1. create a bookmark including the defined term and its definition,
2. in the Glossary section, insert a cross-reference to the bookmark.

Template

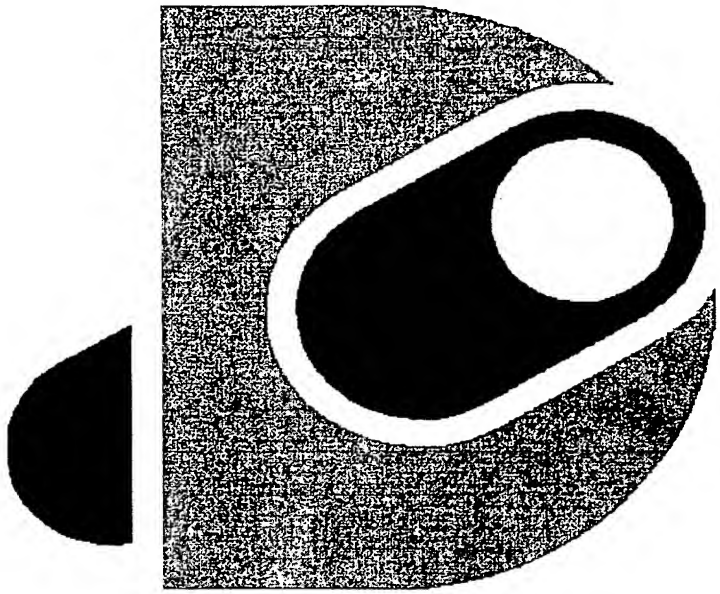
A special kind of document that provides basic tools for shaping a final document.

Index

T

Title : The DataCHEST Ind. ree: Field Topics and Data Objects
Created : 2000-02-11 15:02 By : Alan Lawee
Last modified : 2000-02-11 15:20 By : Alan S. Lawee
Path & Filename : \\BASERVER1\B&A\CLIENTS-C-E (03-05)\0430-DATACHEST COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\INDEX TREE ENTRIES CLEAN.DOC

The DataCHEST Index Tree: Field Topics and Data Objects



DataCHEST.com

Abstract

This document describes the various types of entries contained within the DataCHEST Index Tree.

Field Topics and the DataCHEST Index Tree

Due to the extremely large number of fields in the DataCHEST repository, we have created a representation that we call a "poly-hierarchical tree-structured index", or Index Tree in order to organize the different data fields stored within the repository. (For a technical explanation of the DataCHEST Index Tree, please refer to the document entitled DataCHEST Query Wizard.)

The screen-capture below is a representation of a portion of the DataCHEST Index Tree.

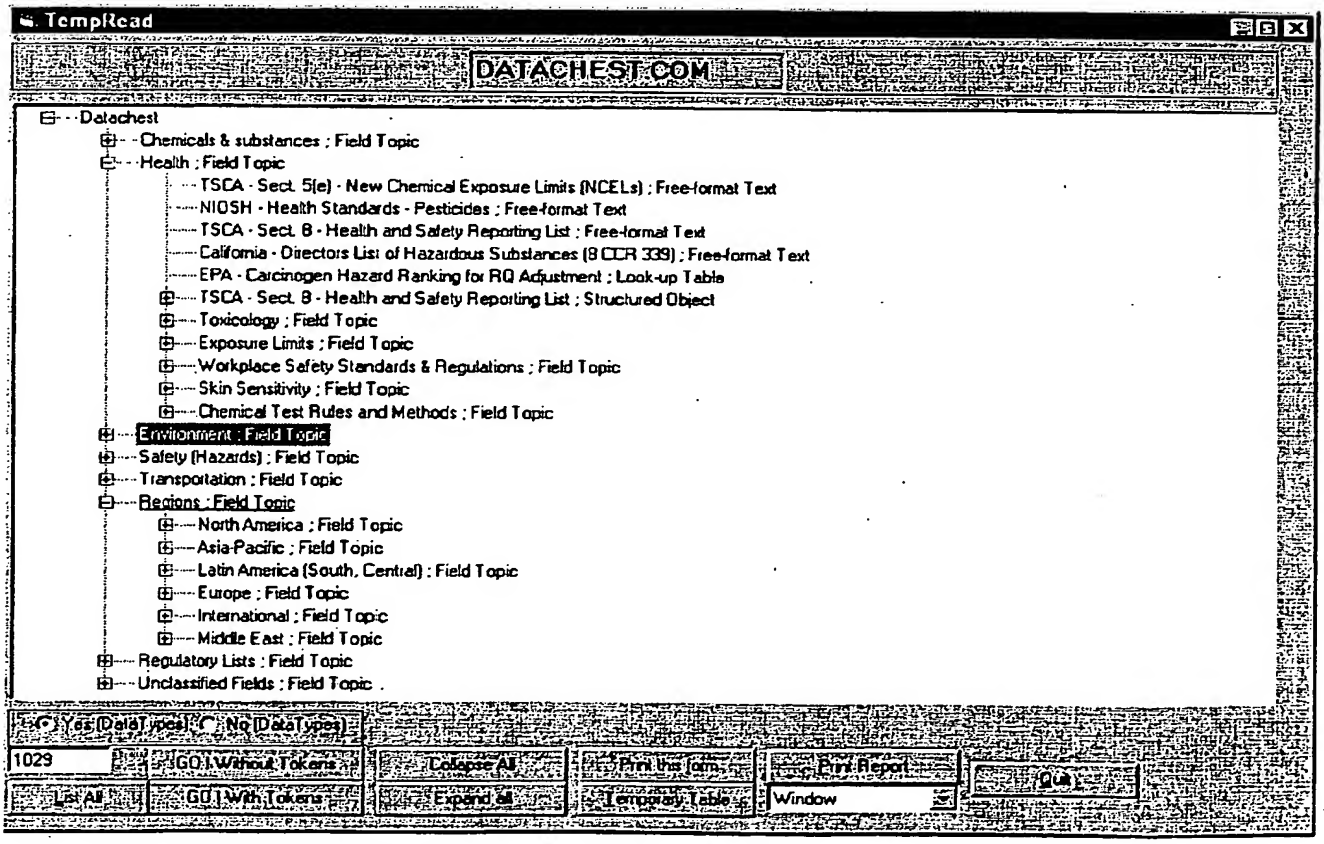


Figure 1 : Screen Capture of a section of the DataCHEST Index Tree.

The entries in the Index Tree can be categorized into three main groups:

1. Field Topics,

2. Data Fields, and
3. Structured Objects.

Field Topics

Field Topics are used to classify and organize the entries within the tree. An appropriate analogy would be to consider them like sub-directories on a computer's hard drive, or folders in a filing cabinet. In Figure 1 above, under the DataCHEST root, the first five Field Topics represent the CHEST acronym : Chemicals, Health, Environment, Safety, and Transportation.

Field Topics can themselves be contained within other field topics. In Figure 1 above, under the Field Topic *"Regions"*, the principal economic continental areas are listed, together with a heading for International, covering data fields that relate across political or economic boundaries. Figure 2, below, displays the same field topic, *"Regions"*, but it is now expanded to show the sub-topic *"Europe"*, with two members, *"European Union"* and *"United Kingdom"*. Below the topic *"European Union"*, some of the member countries are listed. If one of the member countries were to be expanded, the Data Fields classified under that country would appear, together with any further field topics that might apply, such as a state, province, region or city.

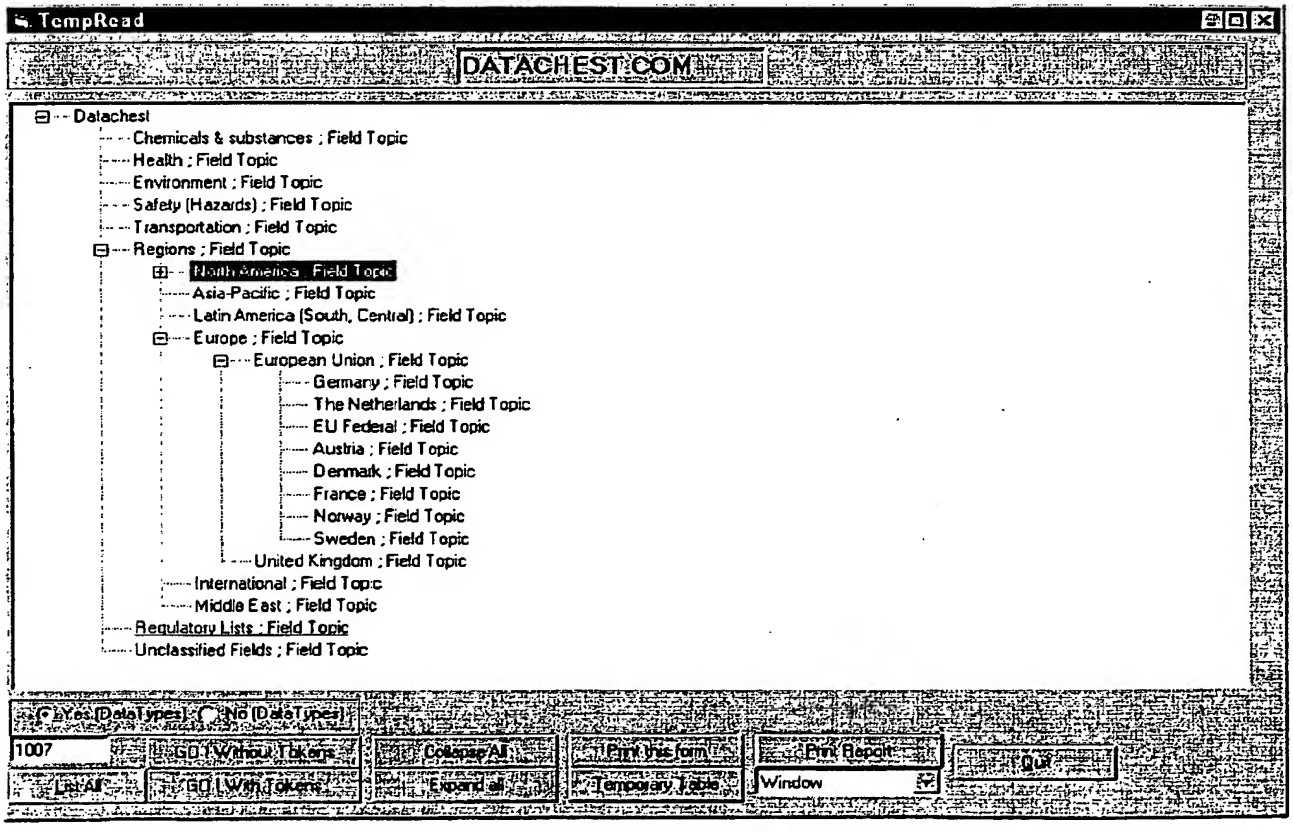


Figure 2 : DataCHEST Index Tree showing expanded Field Topics under the *"Regions"* category.

Data Fields

Data Fields represent the actual data stored within the repository. In Figure 1 above, under the topic *DataCHEST – Health*, there are several examples of Data Fields, such as "TSCA Sect. 5(e) – New Chemical Exposure Limits (NCELs)", "NIOSH – Health Standards – Pesticides", and "EPA – Carcinogen Hazard Ranking for RQ Adjustment".

Data Fields are extremely varied. Although the process of database integration has only just begun, we have already identified over a dozen different data types, and it is expected that more will be added as additional databases are incorporated into our repository. The Data Types currently defined are as follows :

Boolean	Simple binary values (True/False, Yes/No, Present/Not-Present)
Formatted Text	Text value formatted according to a set of rules
Free-format Text	Text value with no structure
Hyperlinks	Link to an object outside of the substance-related database
Image	A graphic image or file, e.g. bit-map, JPEG, GIF, TIFF, etc.
Integer	A whole number (no decimal places)
Large Text (Memo)	Text value whose length exceeds 255 characters
Look-up Table	An index into a table of valid/possible values
Numeric	A numeric value associated with a defined unit of measure
Substance Composition	A special construct for specifying the composition of a substance
Other	A catch-all for otherwise unspecified data types
Field Topic	
Structured Object	

For simplicity of design, Field Topics and Structured Objects are defined in the same Field Names and Field Description tables as the other data types. They can simply be considered as data types that have no data attached.

Structured Objects

Structured objects were devised in order to group together those data elements that are related to each other by more than their related Substance. For example in Figure 3 below, *"Toxicity Test"* results are characterized by a number of individual data elements such as *"Test Subject Animal/Species"*, *"Exposure Period"*, and *"Test Result Description"* that, individually, have very little significance. Taken as a 'set' of data, however, these data elements comprise a representation of the test results.

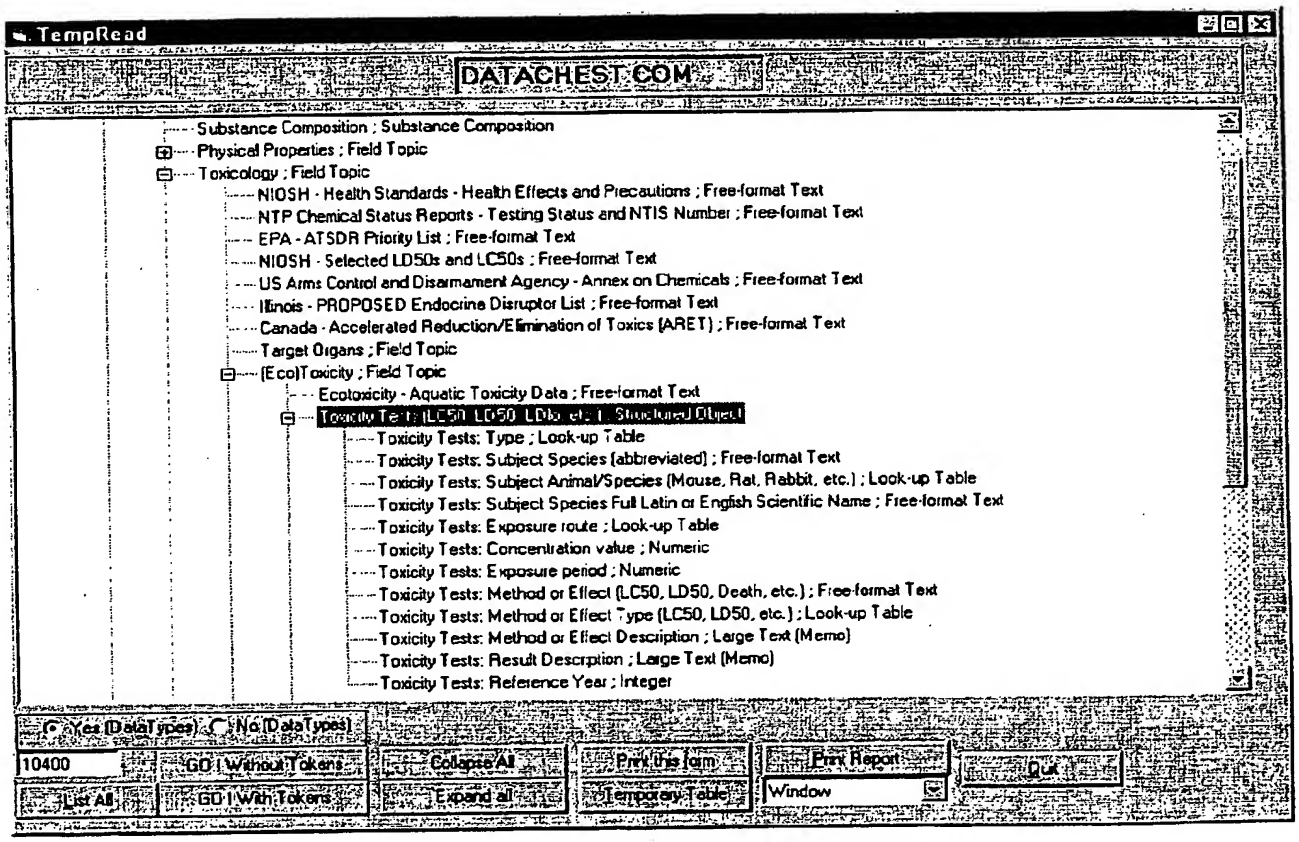
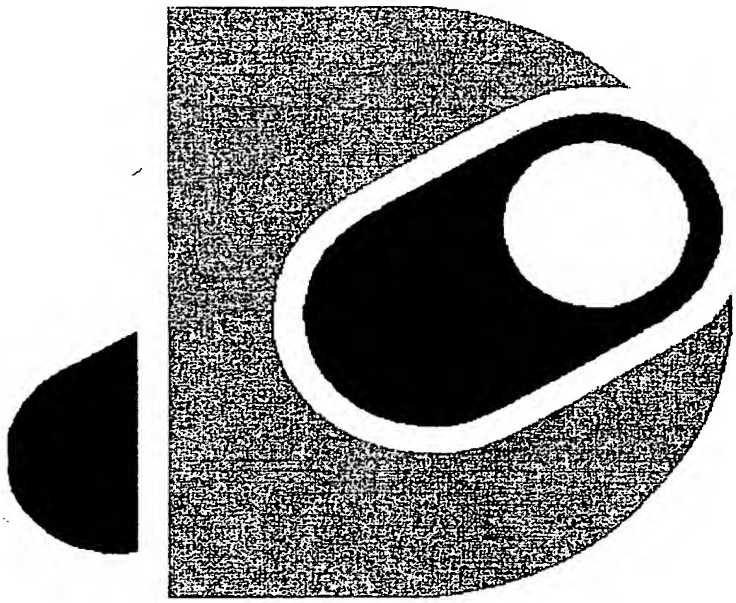


Figure 3 : DataCHEST Index Tree showing a Structured Object.

Title : DataCHEST Database ...apping Utility Design Specifications
Created : 2000-02-11 15:02 By : Alan Lawee
Last modified : 2000-02-11 15:22 By : Alan Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\INDEX TREE UTILITY DESIGN SPEC CLEAN.DOC



DataCHEST.com

007

DataCHEST Index Tree Maintenance Utility Design Specifications

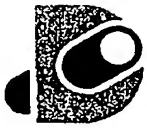
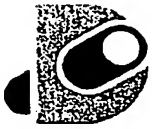


Table of Contents



Abstract

This document summarizes the design specifications for the program that will be used to update and maintain the tables that comprise the DataCHEST Index Tree and Field Mapping structures. This program is intended to be used by the Source Database Mapping Team. This team is made up of Chemical, Health & Safety, Environmental Science, Transportation, Regulatory and/or other industry specialists, who are not necessarily well versed in database architecture and maintenance.

Objectives

The utility is required to facilitate the manipulation and update of the Index Tree Data Structure. The Index Tree is the central structure used to access information within the DataCHEST Repository. Because of the technical nature of the information stored within the repository, it is necessary for the Index Tree to be maintained by personnel with a strong background in Chemistry, Toxicology, Environmental Sciences, Transportation requirements for hazardous materials, and other regulatory fields.

Such personnel are not often skilled in database technology and administration. They must therefore be provided with a simple, intuitive utility that will allow them to add fields, structured objects and field topics to the Index Tree, and to create and/or modify subject groupings and categories, and to classify fields within these groups.

The process of maintaining the Index Tree involves the analysis and study of the source database, the mapping of the source data to appropriate fields within the DataCHEST Repository, creating them where necessary, and specifying data transformations to extract unitary data from composite fields within the source database. A rich library of appropriate topic icons will be available to the mapping team so that they may associate the icons with the fields and topics that are added to the Index Tree, and improve the ease of use of the user interface.

The DataCHEST Query Wizard must provide an on-line documentation and audit of all of the information available in the DataCHEST Repository. This documentation is a natural by-product of the mapping process. Therefore, the Index Tree Maintenance Utility should also update and maintain the Field_Mapping Table in which the documentation data is stored.

Finally, as part of the process of maintaining and creating field entries in the DataCHEST Index Tree, the mapping team must create and maintain entries in the Look-up Table, which defines the set of possible valid values for the contents of data fields assigned to this data type. This function must also be incorporated into the

Table Structures used and updated by the Mapping Utility

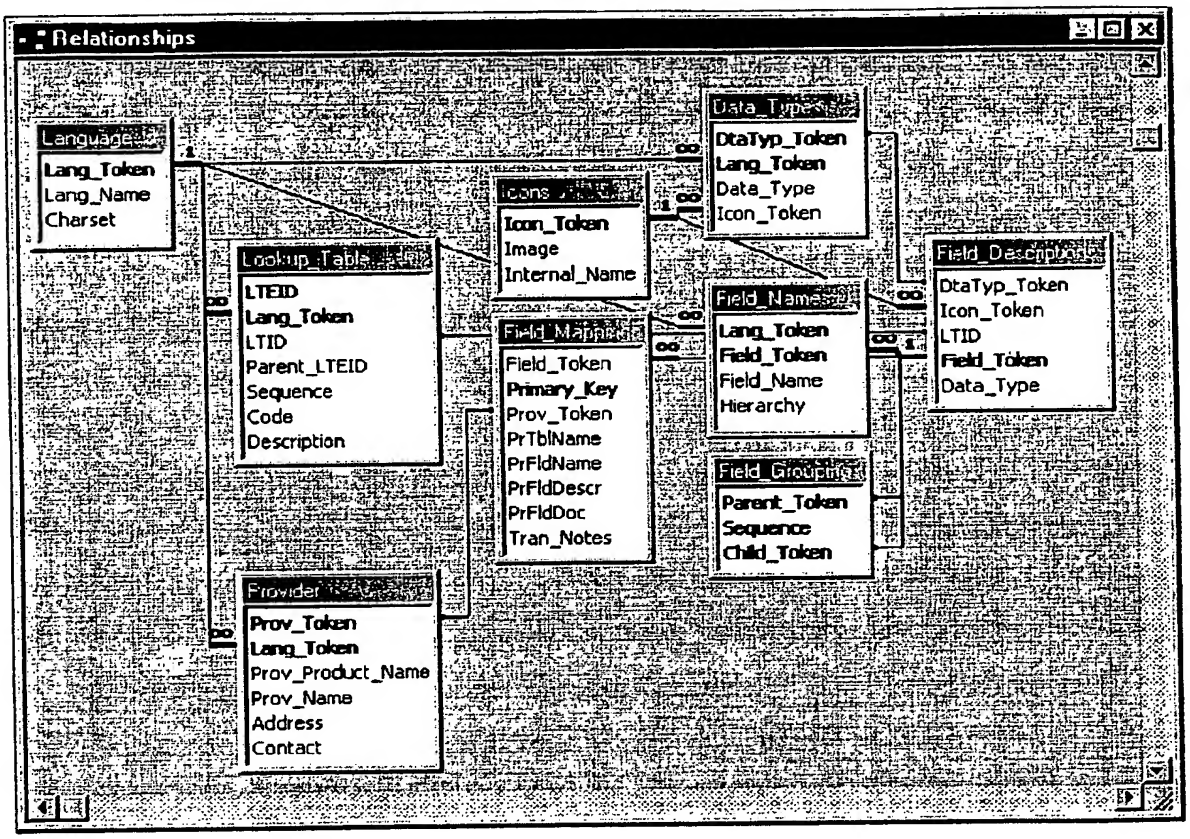
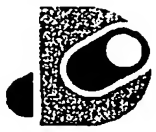


Figure 1: Table Structures within the DataCHEST Repository Schema involved in the Index Tree Maintenance Utility



Program Flow Analysis

Modules:

1. ***Look-up Table***

2. ***Look-up Table Entries***

3. ***DataTypes***

4. ***Icons***

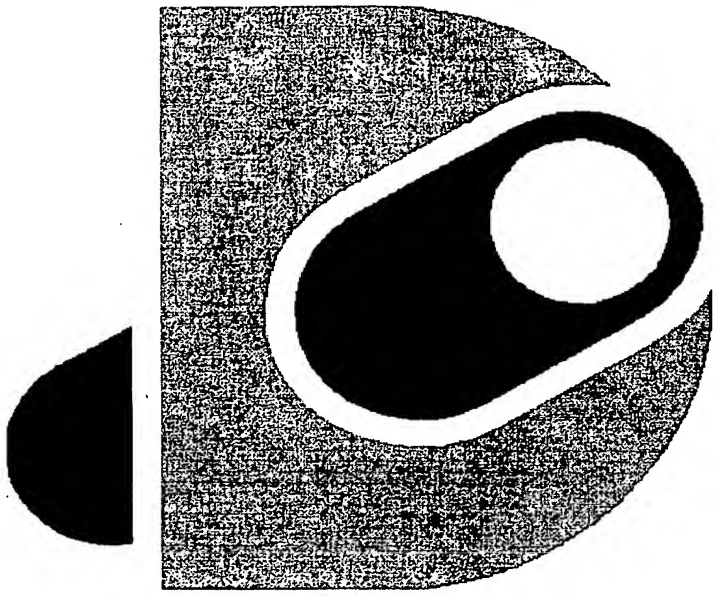
5. ***Field Definition***
(Field_Names, Field_Descriptions)

6. ***Field Grouping***

7. ***Providers***

8. ***Field Mapping***

Title : DataCHEST Query Wizard – Query Technology
Created : 2000-02-11 15:02 By : Alan S. Lawee
Last modified : 2000-02-11 15:25 By : Alan S. Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\QUERY TECHNOLOGY CLEAN.DOC



DataCHEST.com

DataCHEST Query Wizard -- Query Technology

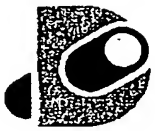
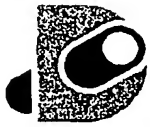


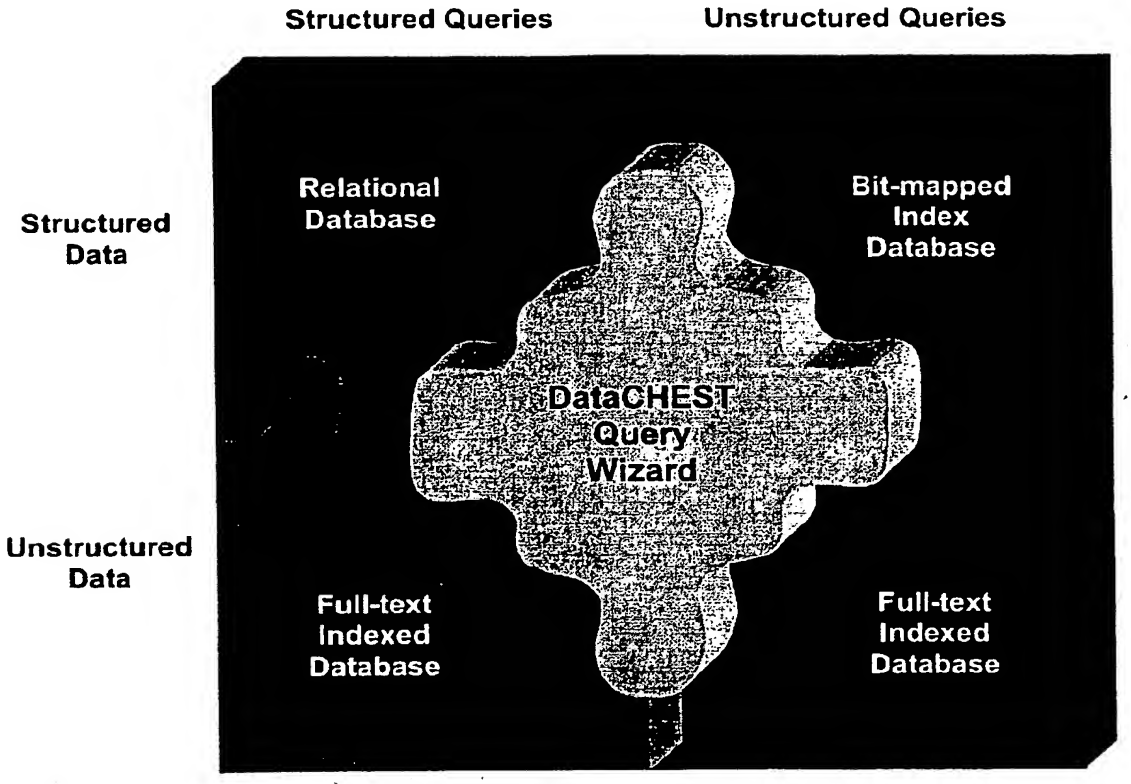
Table of Contents



Abstract

This document summarises the technology and applications that have been considered for use with the DataCHEST Query Wizard.

Query Technology



Query Types and Data Types

Most database queries can be divided into two types, **Structured Queries** and **Unstructured Queries**. **Structured queries** are those that have been anticipated and planned for by the database designer, for example, querying accounts receivable information by asking for the customer by Customer ID number or name. The database designer plans for these queries by indexing the database on the query criteria in order to speed up the retrieval of the relevant information. The usual technology employed to support structured queries is a **relational database**, such as Oracle, Sybase, Informix, MsSQL, Access, and others.

Unstructured queries, on the other hand, are unanticipated ones that do not follow predefined procedures, and whose retrieval process may entail lengthy searches through every item in the database. Typical examples of these are queries on historical sales data that attempt to establish trends according to geographical location, demographic criteria, or other elements that, because of space penalties imposed by the relational databases commonly used to store these types of transaction logs, are not already indexed for speedy retrieval.

To better manipulate **unstructured queries**, a new type of database indexing technique has emerged in the marketplace that uses a **bit-mapped index** structure. Three examples that we have reviewed are Oracle/8, SyBase/IQ, and Nucleus.

Just like queries, data can be characterized into two types, **Structured data** and **unstructured data**. **Structured data** is organized into individual elements, and cross-referenced by the different values that are related to them. **Structured data** can also be categorized into different standard formats, which make it easier to manipulate the data elements.

Unstructured data, on the other hand, is not organized into an ordered and defined structure. This does not mean that it is necessarily disorganized, just that it is not laid out in tabular form. The relationships between the various data elements are based more on context or proximity, rather than on ordered lists and categories. The most common type of **unstructured data** is a collection of documents

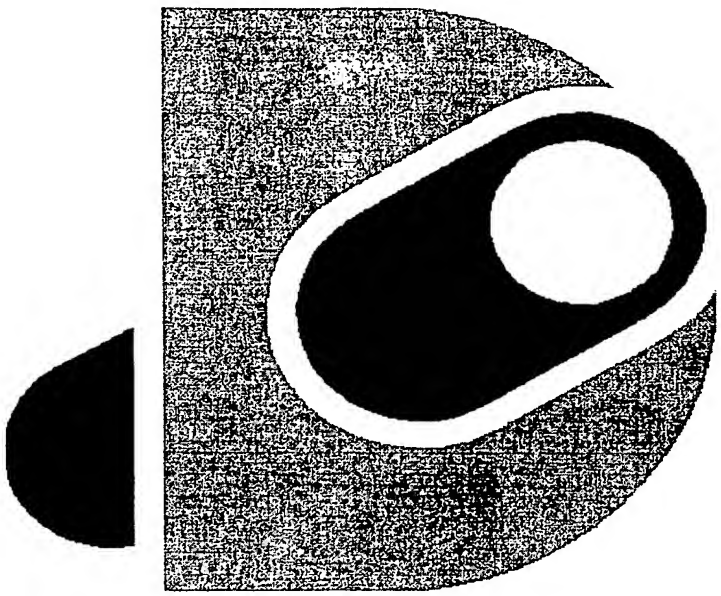
In order to be able to retrieve meaning from **unstructured data**, yet another technique is required. Commonly referred to as **full-text indexing**, it is based on creating a positional index of every word of each document in a collection. Anyone who is familiar with current Internet technology has undoubtedly used search engines, which are the most common examples of this technology.

Each of these three indexing techniques, **Relational**, **Bit-mapped**, and **Full-text**, exist primarily as separate tools (with the possible exception of Oracle 8). Although the industry is moving towards integrating them within a single application, the current state of the tools does not yet permit a seamless interface within a single, off-the-shelf application. The **DataCHEST Query Wizard** incorporates the necessary intelligence to not only utilize the most appropriate technique for each query, but also to mix requests across all three techniques.

For example, it would be possible to import a list of CAS numbers of substances from which the user would like to select only those whose MSDSs have the text *"In case of contact with eyes, flush liberally with water"* located in the First Aid section, whose boiling point is less than 40°C, and for which the reportable quantity threshold is greater than 30,000 metric tonnes usage per year.

Title : DataCHEST Query Wizard
Created : 2000-02-11 15:02 By : Alan S. Lawee
Last modified : 2000-02-11 15:29 By : Alan S. Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\QUERY WIZARD CLEAN.DOC

DataCHEST Query Wizard



DataCHEST.com

Multi-threaded Index Tree and Node Controls

DataCHEST.com

Title: DataCHEST Query Wizard

2000-02-11 15:29

Table of Contents

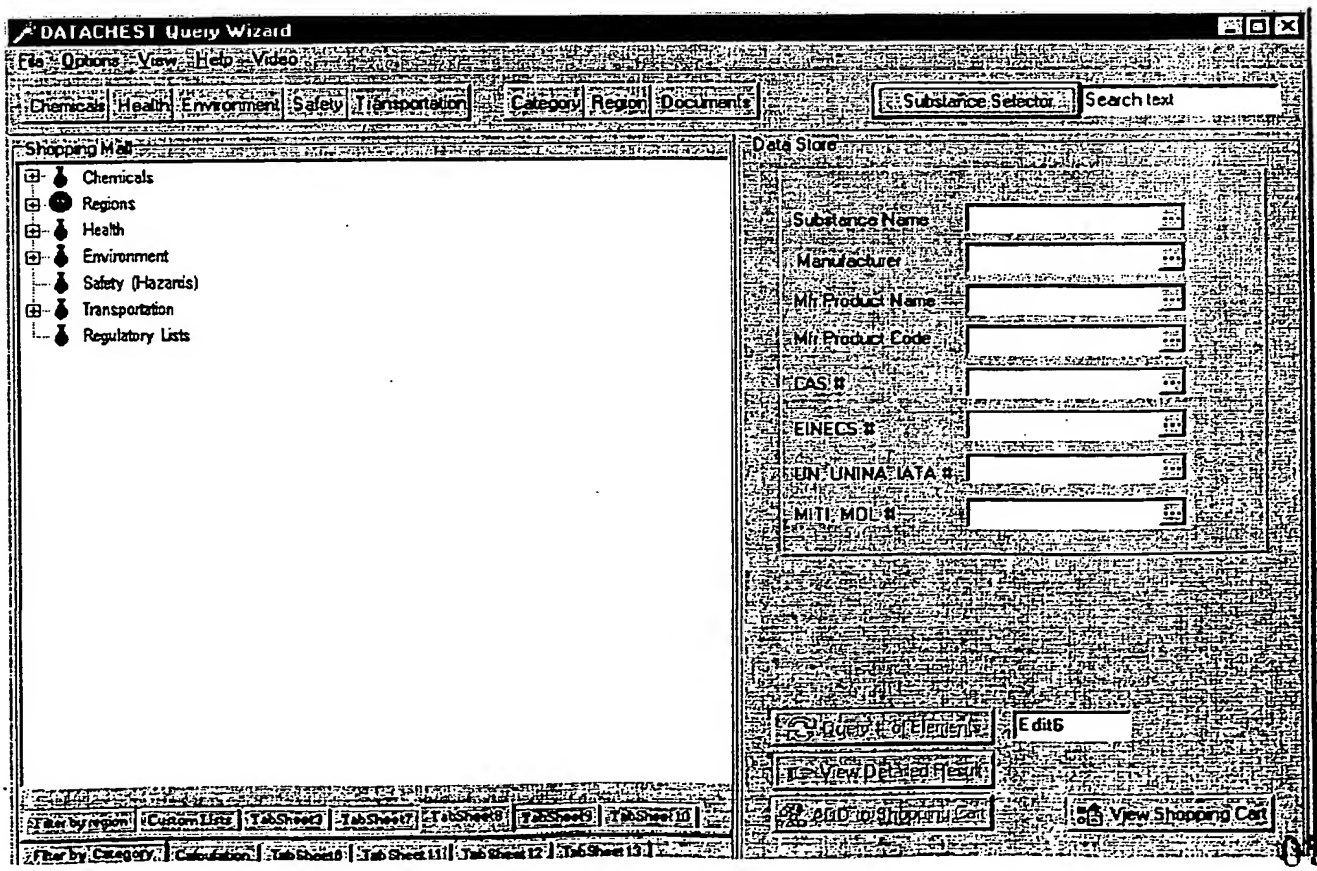
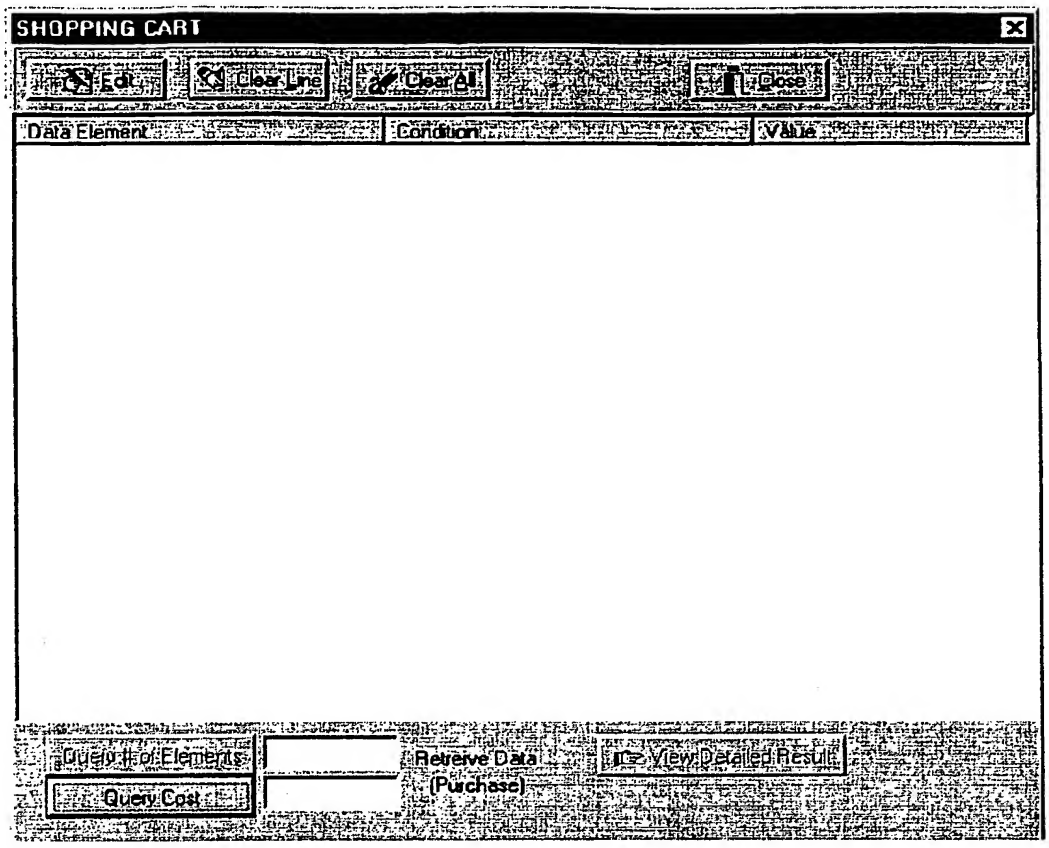


Figure 2 - DataCHEST Query Wizard - Sample Screen Image – Request Buffer



Multi-Threaded Index Tree

(The DataCHEST Shopping Mall)

You're looking for information. Your friends are no help and you've misplaced your library card. What to do ? Get into your NetMobile and click your way to the DataCHEST Information Shopping Mall. Park your car and walk in. What an array of choices you find in front of you ! Where will you begin ?

The Multi-threaded Index Tree concept is the fundamental building block under the DataCHEST Query Wizard. It is a new and unique synthesis of two common Graphical User Interface programming controls: Index Tabs and the Tree control.

The design goal for the DataCHEST Query Wizard is the creation of an intuitive, easy to use indexed catalogue of data element descriptions contained within the DataCHEST data repository. Since it is expected that there will be several thousand distinct data element descriptions and topic headings, the challenge that must be overcome consists of navigating across the tree in a simple fashion that follows the users' thought processes as they formulate their queries.

The Tree control is familiar to everyone who uses a file/directory manager (e.g. Windows Explorer), and thus provides a convenient method to navigate across each level of the tree. The myriad of data element descriptions can be categorised into groups, with either a parent/child connotation, or a topic/element relationship.

The multi-threaded aspect of the tree structure is used to enable the user to follow multiple selection paths to arrive at the same data element description. For example, as shown in the table below, the topic «Toxicology» can fall under the topic headings «Chemical Properties» and «Health» with equivalent justification. Thus we are able to enter multiple relationships into our grouping database in order to represent the topic/sub-topic relationship.

- Root
  - Chemicals
    - Physical Properties
      - Physical Properties - Molecular Formulas
      - Physical Properties - Structural Formulas
      - Physical Properties - Molecular Weights
      - Physical Properties - Water Solubilities
      - Physical Properties - pH Values
    - Toxicology
      - Carcinogens
        - ACGIH 1997 - Notice of Intended Changes - Carcinogens
        - United Kingdom - Carcinogens
        - IARC - Group 1 (carcinogenic to humans)
        - IARC - Group 2A (limited human data)
        - IARC - Group 2B (sufficient animal data)
      - Target Organs
        - NIOSH 1997 - Pocket Guide - Target Organs
- Health
  - Toxicology
    - Carcinogens
      - ACGIH 1997 - Notice of Intended Changes - Carcinogens
      - United Kingdom - Carcinogens
      - IARC - Group 1 (carcinogenic to humans)
      - IARC - Group 2A (limited human data)
      - IARC - Group 2B (sufficient animal data)
    - Target Organs
      - NIOSH 1997 - Pocket Guide - Target Organs
  - Exposure Limits
    - TSCA - Sect. 5(e) - New Chemical Exposure Limits (NCELs)
    - Australian Exposure Standards – Carcinogens

Table 1 - Sample Index Tree showing repetitive Branch Segments under multiple Topics

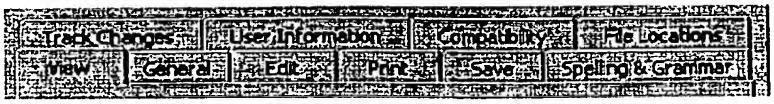
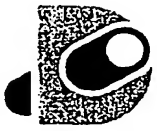


Figure 3 - Sample Tab Control (taken from Microsoft Word)

The tab control toolbar enables another aspect of multi-threading within the index tree. Each root topic can also be considered as a filter, and it should thus be possible to apply multiple filters to the selection process. As the users descend the tree, the contents of the tab control window change to indicate the remaining available filters (root topics). Users can refine their search by clicking on a tab, which will cause the selected root topic to appear below the currently opened tree node. The users can then continue navigating down the tree within the filtered set of data element descriptions.



Data Element descriptions within the Index Tree

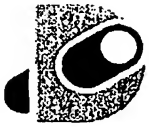
Another critical aspect of the Index Tree is the specification of the individual data element descriptions. This is primarily a manual operation, one that will require great care and discipline.

As DataCHEST is consolidating data from a wide variety of sources, there will inevitably be a fair bit of overlap between the data supplied by the different providers. Unfortunately, not all of the overlapping data will be present in compatible formats, and it will be difficult, in certain instances, to convert the data into common forms that will permit direct comparison of values between providers.

The Data Element descriptions will thus have to reflect the format in which the data is stored. For instance, in the case of the property node for boiling points, there could be several child nodes indicating the format of the data as shown in the example below:

	Sample values
Physical Properties	
Boiling Point	
Text form, units specified	100°C at 760mmHg
Numeric form, degrees Celsius at std. pressure	100
Numeric form, deg. Kelvin at 1.25 atmospheres	375
Object form, units and pressure specified	
State change (text token)	Boils (vs. Decomposes, explodes...)
Temperature (numeric)	100
Unit of measure (text token)	degrees Celsius
Pressure (numeric)	760
Unit of measure (text token)	mm Hg
Remarks	over open flame (vs. Microwave oven)
Melting Point ...	

Selecting a parent node, such as **`Object form`** or **`Boiling Point`**, would return all of the child values associated with it.



Node controls

(Top Data Store)

[illegible]

Once a data element node has been selected, the user will have the possibility of either retrieving all of the data associated with that node, or creating a selection criterion for the selected data element. Much like the User Interface for Microsoft Windows™ Explorer, as each node of the left-pane tree is selected, a custom control is displayed in the right windowpane.

Common control panes would include selection boxes for numeric values and text searches, although many other custom controls will also be created for the DataCHEST Query Wizard application.

A special Substance Selector control would be created to select substances according to

- name, whether a chemical name, brand name or other type of synonym,
- governing body registration number, such as CAS (Chemical Abstract Service), EINECS (European Inventory of Controlled Substances), RTECS (Registry of Toxic Effects of Chemical Substances) or other organisation,
- manufacturer and manufacturer's product code number,
- presence of an ingredient, possibly within a concentration range, or,
- any other form of identifying characteristic.

A means will also be provided whereby the user can upload previously saved or automatically generated lists of substance identifiers.

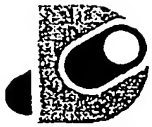
We will provide for custom panes for building news group profiles and for selecting consultants.

Another custom control pane that we envision for inclusion in subsequent releases of our application would entail a graphic selection tool in which the user could build a structural representation of a molecule or sub-structure, and search for a complete or fragmented match.

Additional control panes can be added as necessary. However, every attempt should be made to work within an established set of controls so that confusion and training requirements are kept to a minimum.

As each property node is selected, the Query Wizard can display a count of the number of data elements for that particular node, for instance, if we select Boiling Point, the application should display the fact that the Repository contains information on 5000 substances. Displaying this type of information is considered to be good public relations, and there should not be a charge for it.

On the other hand, displaying a filtered result can often have a definitive value and should therefore be chargeable. It is one thing to publicise that our Database contains boiling points for 5000 substances, and quite another to qualify that only 300 of these fall below -33° C



Request Buffer

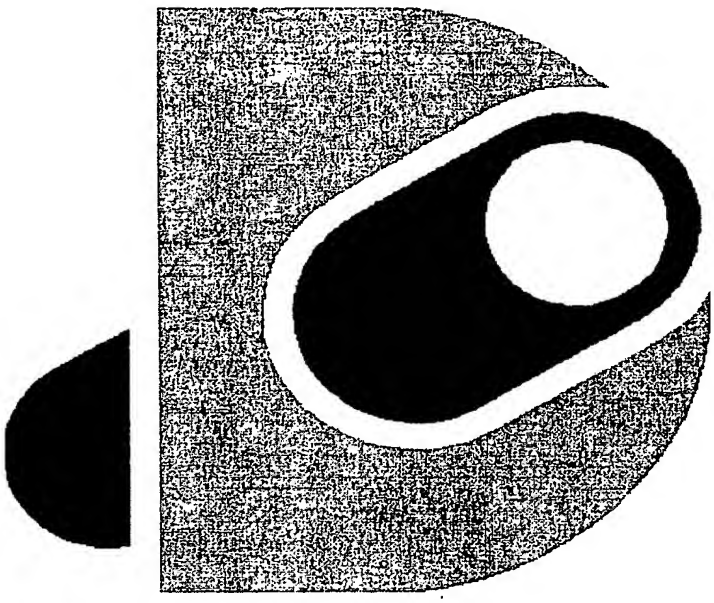
(The 'Shopping Cart')

That's just what you've been looking for. Now let's put it into the shopping cart so that once we have found everything on our list, we can take it to the checkout counter and purchase it !

As each criterion is completed, it is added to the request buffer. Several criteria can be combined into a single request, and, using the Boolean operators AND, OR, NOT and XOR, the request can be as general or as specific as needed. The user should have the necessary tools to modify the criteria making up a request at his discretion. Pressing the «**Query Cost**» button at any time should perform the request without retrieving the data, and display the net cost to the user, after applying the appropriate discounts. We are considering providing this information at no charge to the user.

Querying the number of hits for a query could actually be useful information, and it is thought that a small charge would apply when the «**Query # of Elements**» button is pressed.

Once the user has decided upon a query, he or she would proceed with the transaction by pressing the «**Retrieve Data (Purchase)**» button. After confirming the request and accepting to pay the quoted amount, the request would be re-executed (hopefully from cache) and the result set returned to the client in the requested format (HTML or JDBC data set). The Client would then be able to view and manipulate the data at his discretion.



DataCHEST.com

Result Presentation Notes

Table of Contents

Abstract

This document covers interim modifications to the presentation of the Results Page of the Query Application.

Modifications required for the Results Page presentation

Display Structured Objects as a sub-group within the result block.

Currently, all data nodes are displayed in somewhat random order in the result block (group-by). If there are two 'hits' for the same SFID and Version_Token, they are grouped within the same field cell and displayed in random order.

This is fine for independent fields (children of Field Topics), but is disastrous for dependent fields (children of Structured Objects).

We must come up with a way around this problem on the Results Page. Ideally, a Structured Object Parent Field should be displayed exactly as an independent field, and its children should be displayed to its right in separate rows. I have given an example below:

Substance Name	Water	
Substance CAS number	000-00-0	
Information Version	DataCHEST Sample:v1;DataCHEST Sample Data:v1;	
Information Provider	DataCHEST.com Inc.	
Independent Field (e.g. Boiling Point)	= 212 °F	
	= 100 °C	
Structured Object (e.g. Arsenic Levels in drinking water)	Dependant field (e.g. Arsenic concentration %)	< 0.1 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Not harmful
	Dependant field (e.g. Arsenic concentration %)	< 5.0 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Somewhat harmful
	Dependant field (e.g. Arsenic concentration %)	> 10.0 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Fatal

Results display: Example of a structured object.

Please find below an example of the results table taken from a sample query requesting TSCA Health & Safety information for substances whose name begins with "Toluen". Four Structured Objects were selected for "Display" in the dynamic filters. These were:

- U.S. EPA - TSCA - Sect. 8 (d) - Health and Safety Reporting List
  - U.S. EPA - TSCA - Sect. 8 - Health and Safety Reporting List - Effective date
  - U.S. EPA - TSCA - Sect. 8 - Health and Safety Reporting List - Reporting date
  - U.S. EPA - TSCA - Sect. 8 - Health and Safety Reporting List - Expiry (sunset) date
- U.S. EPA - SARA - 312 - Chronic health hazard List
  - U.S. EPA - SARA 312 - Chemical listed as a chronic health hazard - (Yes/No)
  - U.S. EPA - SARA 312 - Category for chronic health hazard
- U.S. EPA - TSCA - Sect. 8 (d) - Health and Safety Studies Reporting List
  - U.S. EPA - TSCA - Sect. 8 (d) - Health and Safety Studies Reporting List (Present)
  - U.S. EPA - TSCA - Sect. 8 (d) - Health and Safety Studies Reporting List - Sunset date
- U.S. EPA - SARA - 312 - Acute health hazard List
  - U.S. EPA - SARA 312 - Chemicals listed as an acute health hazard - (Yes/No)
  - U.S. EPA - SARA 312 - Category for acute health hazard

The relevant fields of the query results table for this request are listed on the next page:

```
SQL> select sfid, version_token, fg$parent_token, parent_deid, fg$dc_sequence, deid
  2  from query_ol_data_elements where QUERY_SUMMARY_NO = 4346
  3  order by sfid, version_token, fg$parent_token, parent_deid, fg$dc_sequence;
```

SFID	VERSION_TOKEN	FG$PARENT_TOKEN	PARENT_DEID	FG$DC_SEQUENCE	DEID
11490	18	264	10431722	2	10431723
11490	18	264	10431722	3	9999993
11490	18	264	10431722	4	10431725
15422	18	264	10431882	2	10431883
15422	18	264	10431882	3	9999993
15422	18	264	10431882	4	10431885
16025	18	264	10432038	2	10432039
16025	18	264	10432038	3	9999993
16025	18	264	10432038	4	10432041
20002	18	264	10432306	2	10432307
20002	18	264	10432306	3	9999993
20002	18	264	10432306	4	10432309
47690	18	264	10433062	2	10433063
47690	18	264	10433062	3	9999993
47690	18	264	10433062	4	10433065
79312	18	264	10433686	2	10433687
79312	18	264	10433686	3	9999993
79312	18	264	10433686	4	10433689
138057	18	264	10434454	2	10434455
138057	18	264	10434454	3	9999993
138057	18	264	10434454	4	10434457
142833	18	264	10434518	2	10434519
142833	18	264	10434518	3	9999993
142833	18	264	10434518	4	10434521
155910	18	264	10434606	2	10434607
155910	18	264	10434606	3	9999993
155910	18	264	10434606	4	10434609
159373	18	264	10434686	2	10434687
159373	18	264	10434686	3	9999993
159373	18	264	10434686	4	10434689
159378	18	264	10434694	2	10434695
159378	18	264	10434694	3	9999993
159378	18	264	10434694	4	10434697
160038	18	264	10434718	2	10434719
160038	18	264	10434718	3	9999993
160038	18	264	10434718	4	10434721
160111	18	264	10434738	2	10434739
160111	18	264	10434738	3	9999993
160111	18	264	10434738	4	10434741
214491	55	22741	16671017	1	5
214491	55	22741	16671017	2	16671018
214491	55	22741	16671019	1	5
214491	55	22741	16671019	2	16671020
214851	55	22741	16672139	1	5
214851	55	22741	16672139	2	16672140
214851	55	22741	16672141	1	5
214851	55	22741	16672141	2	16672142
215091	55	22741	16672283	1	5
215091	55	22741	16672283	2	16672284
215091	55	22741	16672285	1	5
215091	55	22741	16672285	2	16672286
215139	55	22741	16672357	1	5
215139	55	22741	16672357	2	16672358
215216	55	22741	16672359	1	5
215216	55	22741	16672359	2	16672360
215777	55	22741	16671029	1	5
215777	55	22741	16671029	2	16671030
215777	55	22741	16671031	1	5
215777	55	22741	16671031	2	16671032
215783	55	22741	16670977	1	5
215783	55	22741	16670977	2	16670978
215783	55	22741	16670979	1	5
215783	55	22741	16670979	2	16670980
215998	55	22741	16671841	1	5
215998	55	22741	16671841	2	16671842
215998	55	22741	16671843	1	5
215998	55	22741	16671843	2	16671844
221865	55	22741	16671041	1	5
221865	55	22741	16671041	2	16671042
221865	55	22741	16671043	1	5
221865	55	22741	16671043	2	16671044
223034	55	22741	16672235	1	5
223034	55	22741	16672235	2	16672236
223034	55	22741	16672237	1	5
223034	55	22741	16672237	2	16672238
223175	55	22741	16672259	1	5
223175	55	22741	16672259	2	16672260
223175	55	22741	16672261	1	5
223175	55	22741	16672261	2	16672262
272152	55	22741	16671021	1	5
272152	55	22741	16671021	2	16671022
272152	55	22741	16671023	1	5
272152	55	22741	16671023	2	16671024

83 rows selected.

This data should be presented in the manner shown below, where the text descriptions have been replaced with the token values from the above table.

Of the three table blocks below, the first is a structure representation shown with the variable names, and the next two are typical examples using representative sets of data have been extracted from the above data (as shown in Red Text on a Gray Background).

In the case of an autonomous field (not a child of a structured object), the FG$PARENT_TOKEN and FG$DC_SEQUENCE values can be safely ignored (they should be Null in any case), and the PARENT_DEID value should be Null or 0 (zero).

Substance	SFID		
Version	VERSION_TOKEN		
Autonomous Field FIELD_TOKEN	DEID		
Structured Object FG$PARENT_TOKEN		FIELD_TOKEN	DEID

The middle block displays a structured object in a query result that returned a single set of data elements.

Substance	160111				
Version	18				
Structured Object 264		Field_Token	265	DEID	10434739
		Field_Token	266	DEID	9999993
		Field_Token	267	DEID	10434741

The last block displays a structured object in a query result that returned a multiple set of data elements.

Substance	214851				
Version	55				
Structured Object 22741		Field_Token	22742	DEID	5
		Field_Token	22743	DEID	16672140
		Field_Token	22742	DEID	5
		Field_Token	22743	DEID	16672142

[redacted], and are used only for ordering and grouping.

The breakpoints on Substance Form (SFID) and Information Version (VERSION_TOKEN) serve to open a new grouping table. The PARENT_DEID serves to identify the sub-groups for the structured object data sets within the SFID/VERSION_TOKEN group. Lastly, the FG$DC_DEQUENCE serves to order the FIELD_TOKEN/DEID elements within the sub-group.

Addition of Effective Date and Expiry Date fields to Preset Filters

Also, as we discussed this morning, with the new fields (Effective/Expiry Dates) being added to, or corrected in, the Presets, we will have to change the results page once again. We have two choices here:

1.) we can add these fields to the GROUP-BY criteria, as in the following example

Substance Name	Water
Substance CAS number	000-00-0
Information Version	DataCHEST Sample;v1;DataCHEST Sample Data;v1;
Information Provider	DataCHEST.com Inc.
Information Effective Date	15-Nov-1999
Information Expiry Date	31-Dec-2000
Independent Field (e.g. Boiling Point)	= 212 °F
	= 100 °C

Substance Name	Water	
Substance CAS number	000-00-0	
Information Version	DataCHEST Sample;v1;DataCHEST Sample Data;v1;	
Information Provider	DataCHEST.com Inc.	
Information Effective Date	01-Jan-2000	
Information Expiry Date	31-Dec-2000	
Structured Object (e.g. Arsenic Levels in drinking water)	Dependant field (e.g. Arsenic concentration %)	< 0.1 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Not harmful
	Dependant field (e.g. Arsenic concentration %)	< 5.0 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Somewhat harmful
	Dependant field (e.g. Arsenic concentration %)	> 10.0 %
	Dependant field (e.g. Years of daily exposure)	10
	Dependant field (e.g. Degree of toxicity)	Fatal

2.) or, we can add columns to the result table like this :

Field	Data		Effective Date	Expiry Date
Substance Name	Water			
Substance CAS number	000-00-0			
Information Version	DataCHEST.com Inc.; v1; Sample Data; v1;		01-Jan-2000	31-Dec-2000
Information Provider	DataCHEST.com Inc.			
Independent Field (e.g. Boiling Point)	= 212 °F		15-Nov-1999	31-Dec-2000
	= 100 °C			
Structured Object (e.g. Arsenic Levels in drinking water)	Dependant field (e.g. Arsenic concentration %)	< 0.1 %	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Not harmful	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Arsenic concentration %)	< 5.0 %	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Somewhat harmful	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Arsenic concentration %)	> 10.0 %	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000	31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Fatal	01-Jan-2000	31-Dec-2000

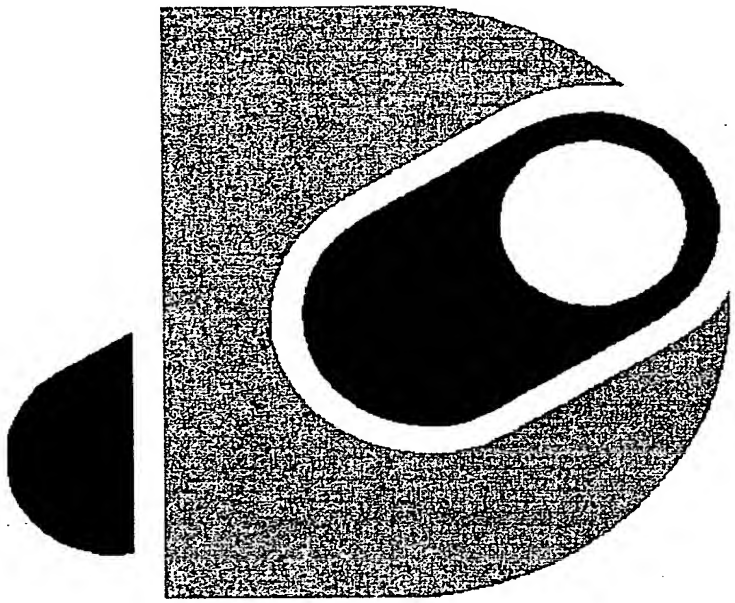
Or like this:

Field	Data		Effective Date / Expiry Date
Substance Name	Water		
Substance CAS number	000-00-0		
Information Version	DataCHEST.com Inc.; v1; Sample Data; v1;		01-Jan-2000 31-Dec-2000
Information Provider	DataCHEST.com Inc.		
Independent Field (e.g. Boiling Point)	= 212 °F		15-Nov-1999 31-Dec-2000
	= 100 °C		15-Nov-1999 31-Dec-2000
Structured Object (e.g. Arsenic Levels in drinking water)	Dependant field (e.g. Arsenic concentration %)	< 0.1 %	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Not harmful	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Arsenic concentration %)	< 5.0 %	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Somewhat harmful	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Arsenic concentration %)	> 10.0 %	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Years of daily exposure)	10	01-Jan-2000 31-Dec-2000
	Dependant field (e.g. Degree of toxicity)	Fatal	01-Jan-2000 31-Dec-2000

Addition of Synonym Type

Another issue that affects the same code in the application is the display of the synonym type information. Because a synonym type is related to a specific name, it must be attached to the name when it is being displayed. Ideally, this would add another sub-cell (column) to the name data, as demonstrated by the following example:

Field	Data	
Substance Name	Water	Common Name
	Laurentian Spring Water	Trade or Brand Name
Substance CAS number	000-00-0	
Information Version	DataCHEST.com Inc.; v1; Sample Data; v1;	
Information Provider	DataCHEST.com Inc.	
Independent Field (e.g. Boiling Point)	= 212 °F	
	= 100 °C	



DataCHEST.com

The DataCHEST Database Schema

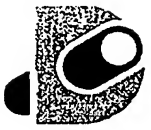
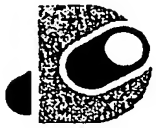


Table of Contents

Abstract

This document describes the design of the DataCHEST.com data repository schema. It attempts to explain the rationale behind the design, as well as the various dimensions of the data contained within the repository.



The DataCHEST Data Repository

The diagram on the following page (Figure 1) displays the database schema for the main relational database within the DataCHEST repository.

One of the initial criteria for the design of the database is the requirement to support all text fields in the database as multilingual fields. This criterion results in some additional complexity within the database. For example, in the Substances table, rather than including a simple text field for "Physical State", it is necessary to include a token pointing to another table containing the text field in the various languages desired. The same logic dictates the inclusion of several other tables, like the Data_Types table, the Field_Names table, and the Synonym_Types table to name a few.

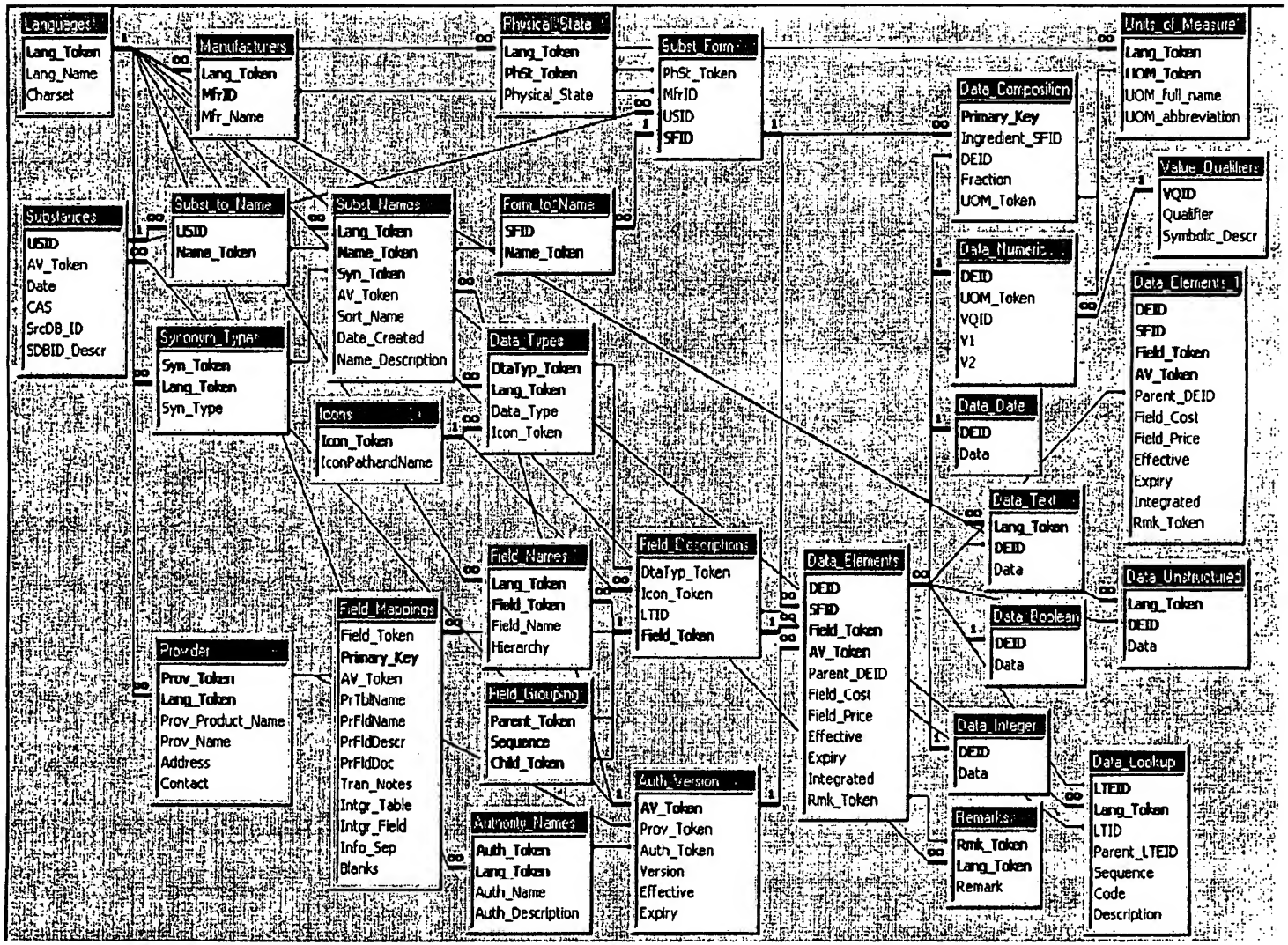


Figure 1 : DataCHEST repository (Substance-related data) database schema, with language relationships shown.

Design Objectives

The Schema becomes considerably less cluttered if the language relationships are hidden, as evidenced by the diagram in Figure 2 on the following page.

The objective for the database design is to achieve a structure in which it is possible to assimilate a large number of source data elements originating within many different source databases, each with its own particular structure, language, and environment.

For the most part, the source data elements all relate to one or more substances, sometimes dependent on the physical state of the substance (solid, liquid, gas, etc.), and also at times dependent upon the source or manufacturer of the substance. Data elements must also be identified by the provider, version and authority of their source.

To represent such a structure as a common two-dimensional tabular matrix would be cumbersome and unwieldy, resulting in thousands of columns of data and hundreds of thousands of rows of substances. The concept of data sources and versions would only add to the confusion.

By converting the tabular column dimension into a virtual 'row' dimension (the Field Dimension, examined below), we are able to more efficiently manage and manipulate the large amount of data in the repository.

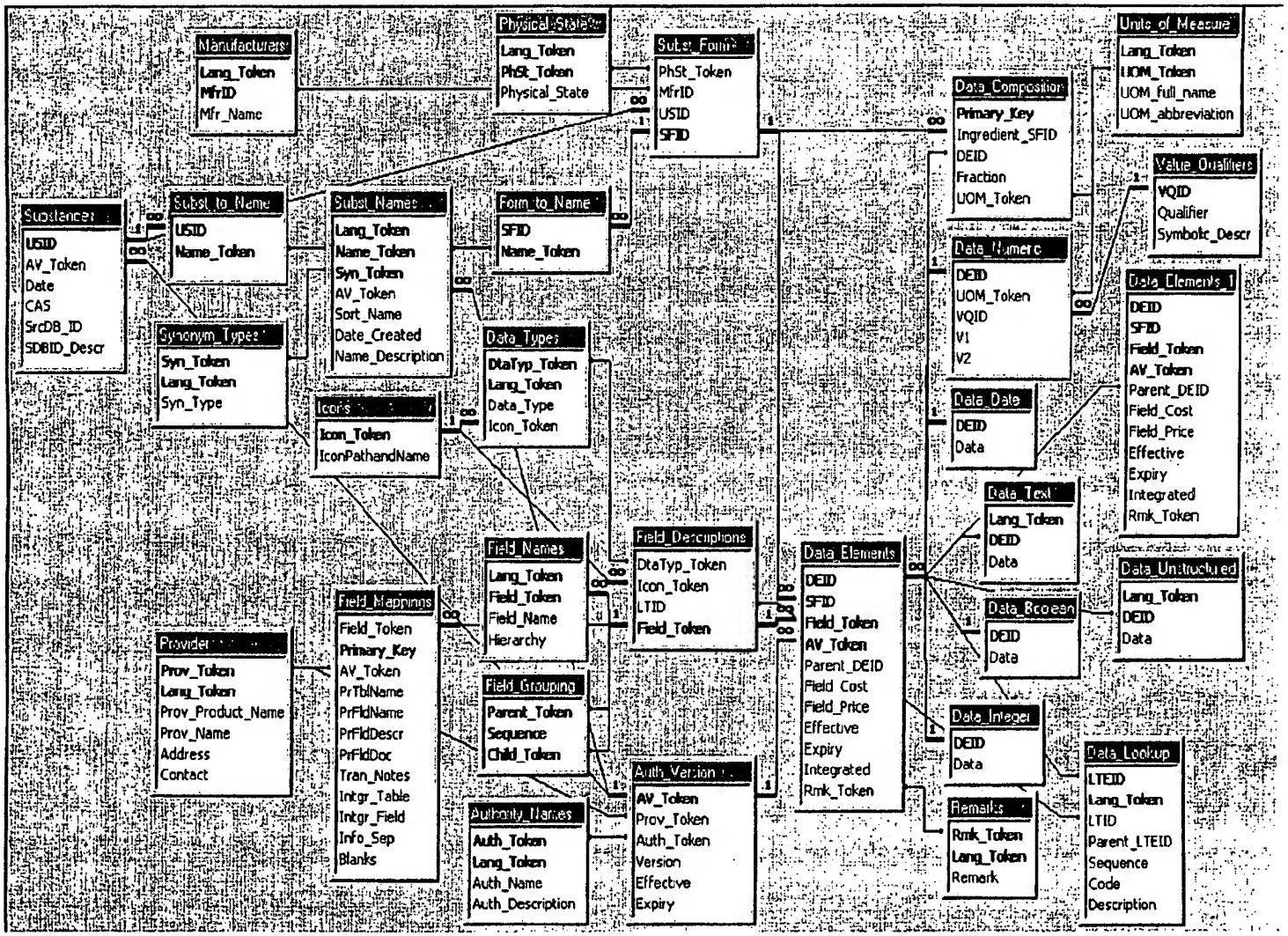


Figure 2 : DataCHEST repository (Substance-related data) database schema, with language relationships hidden.

The Data Elements Table

The strategy employed by the DataCHEST repository is to convert the 'rows', 'columns' and 'data source and version' information into dimensions of a central 'Data_Elements' table, as highlighted in Figure 3 on the following page

Each of the 'Data_Elements' table entries is identified by a Substance (USID, or unique substance identifier), a 'Field_Token' indicating the nature of the data element, and an Authority/Version token (AV_Token) representing the source and version of the data element.

In view of the wide variety of data element types, it would be an extremely inefficient waste of table space to store the actual data in the same table as the dimensional information. Instead, a series of data tables are used to store the data, one for each data type. The data in these tables are linked to the 'Data_Elements' by another token, the Data Element Identifier (DEID).

The nature of the source data is such that a series of data elements occasionally contain entries that include comments or qualifiers describing or limiting the individual value. To accommodate this characteristic, the Data_Elements table is linked to a Remarks_Table via a separate token (Rmk_Token).

In order to support the notion of structured objects, the Parent_DEID field has been included in the 'Data_Elements' table. This will be explained later in the Data Dimension section of this document,

For billing purposes, cost and pricing information is included with each Data Element entry.

Date information is also recorded for effective and expiry dates, as well as for the creation date of the data element.

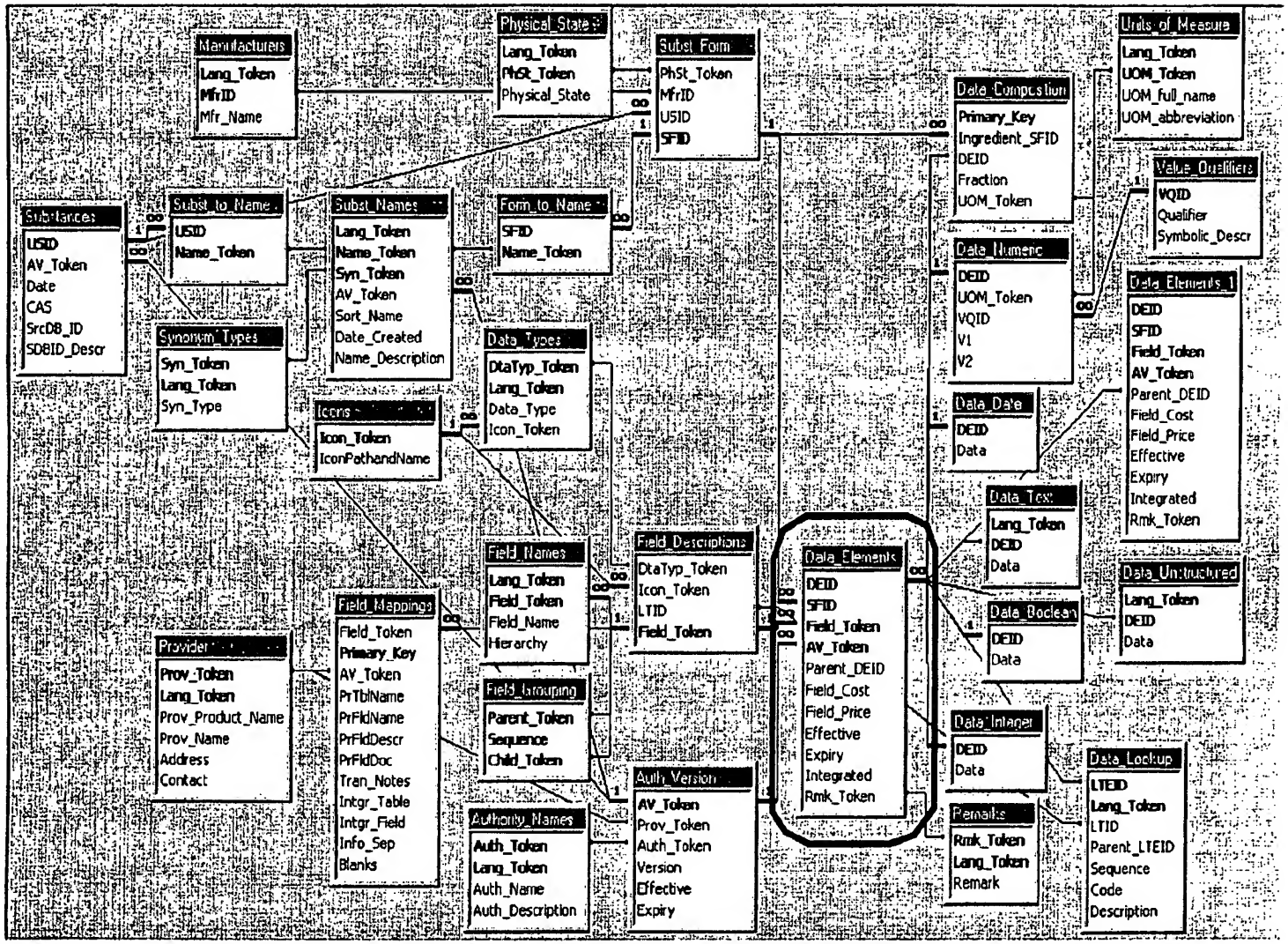


Figure 3 : DataCHEST repository Data Elements Table.

The Substance Dimension

As indicated earlier, the Data_Elements table is related to a Unique Substance Form, qualified by a unique Substance, a Physical State and a source (Manufacturer). Multi-lingual substance names are held in a separate table (Subst_Names). Since a substance can have more than one name, or synonym, every name is characterized by a Synonym_Type, and again, in order to be able to store multi-lingual synonym names, they are, in turn, stored in their own table.

In order to model the many-to-many relationships between Substance Names and Substances and Substance forms, two extra tables, Subst_to_Name and Form_to_Name are used. These tables consist of a Name Token and unique Identifier that is used to link to the Substance and Substance Forms table.

When trying to understand the data model, it is useful to think of a substance, for instance water, which exists in several physical forms, namely ice, water and steam. Water can also originate from different sources, for example, distilled water, Perrier water, Evian water, etc.

Each of these varieties of water can possess attributes that are common across many varieties, such as viscosity or vapor pressure. They can also have properties that are unique to a physical state or source, such as density or salinity.

A single variety of water can also have many synonyms. Take snow, for instance, which can be called powder snow, corn snow, or a variety of other names. The Eskimo language, Inuktituk, uses over 40 different terms to describe snow.

More to the point, various regulatory bodies have catalogued chemical substances for their own internal purposes, and often use their own version of a name to describe the substance. Because these regulatory listings have the force of law, the names used to describe a substance become unique synonyms, even if they differ from one another due to punctuation or even a spelling mistake.

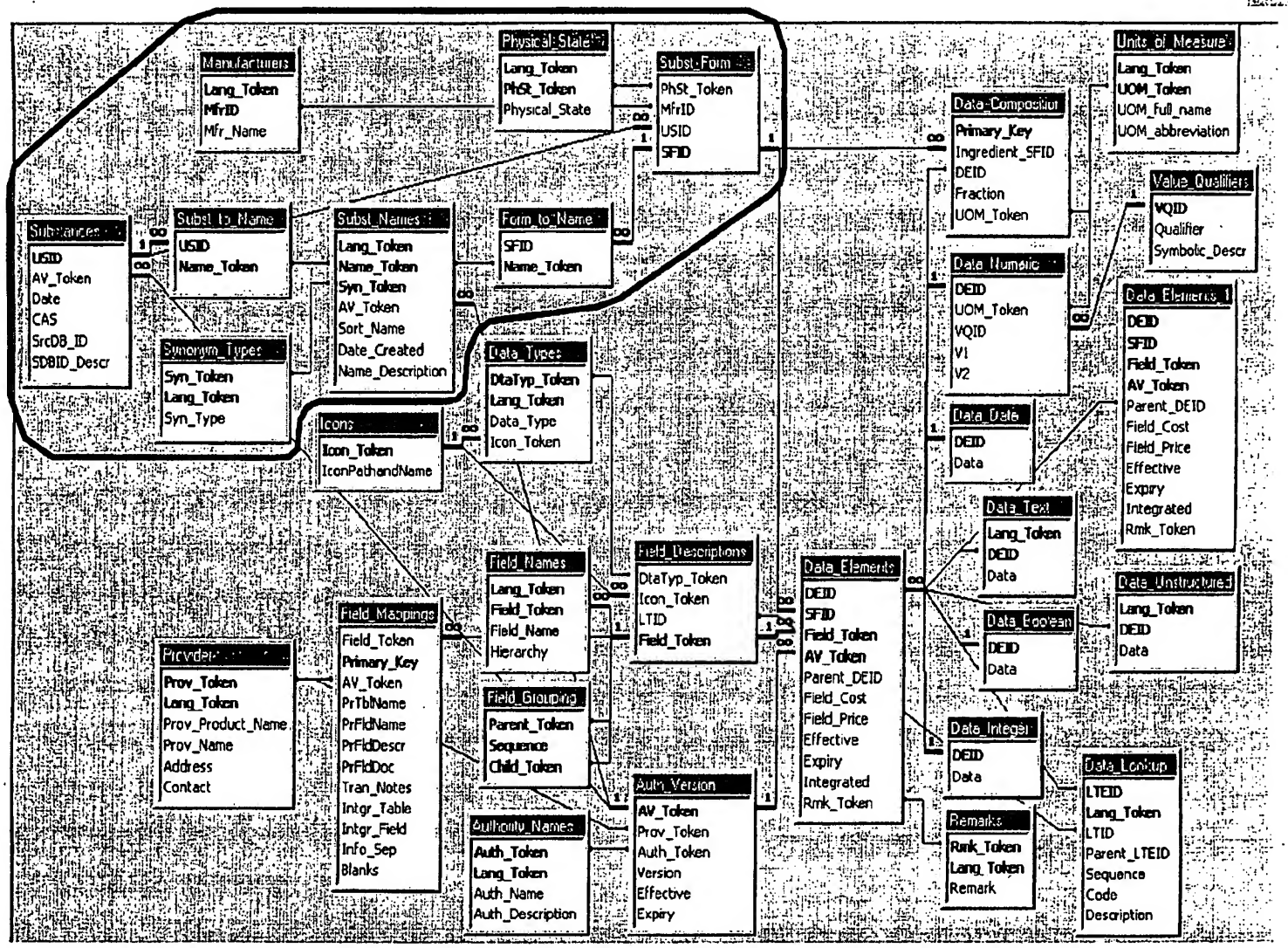


Figure 4 : **DataCHEST repository, Substance-Dimension.**

The Data Field Dimension

The Data Field dimension describes the nature of the data elements within the repository.
Data Fields have two basic attributes, the Field Name and the Data Type. In order to accommodate multilingual naming of Field Names and Data Types these attributes must be stored in separate tables.
In order to permit users to view a Data Field's documentation on-line, a Field_Mapping table has been defined to describe its relationship to the original source data as furnished by the data provider.
Another construct, the Field_Grouping table, was created in order to implement the Query Wizard Index Tree structure. This table identifies pairs of parents and children and an optional sequence number for specifying an ordering for the children.
To enhance the look and feel of the user interface, a Data Field may optionally be associated with an icon, which will be displayed next to it in the Query Wizard Index Tree. Icons are also associated with data types, so a Data Field can thus have two icons associated with it. Because more than one Data Field can apply to the same icon, they have been stored in their own table.
Finally, the Lookup_Table construct has been created to enhance the Index Tree and make data storage more efficient. It is used for data element types where the possible values fall within a pre-determined set of values. For example, IARC, the International Association for Research into Cancer, publishes a list of chemical substances together with their assigned carcinogenicity category, as described in the table below:

LTEID	LTID	Lang Token	Parent LTEID	Sequence	Code	Description
97	10059	1	0	0		IARC (International Agency for Research on Cancer) Carcinogen Classes
98	10059	1	97	1	1	Carcinogenic to humans
99	10059	1	97	2	2A	Suspected Human Carcinogen - Limited human data
100	10059	1	97	3	2B	Suspected Human Carcinogen - Sufficient animal data
101	10059	1	97	4	3	Not classifiable
102	10059	1	97	6	4	Probably not carcinogenic
103	10059	1	97	7	5	Unspecified

The use of the lookup-table construct permits the display of the possible values for the IARC Carcinogen classes, together with a more detailed explanation of their meaning. By implementing the lookup table as part of the Data Field dimension, the values can be displayed as part of the Index Tree and as part of the selection criteria drop-down tables.

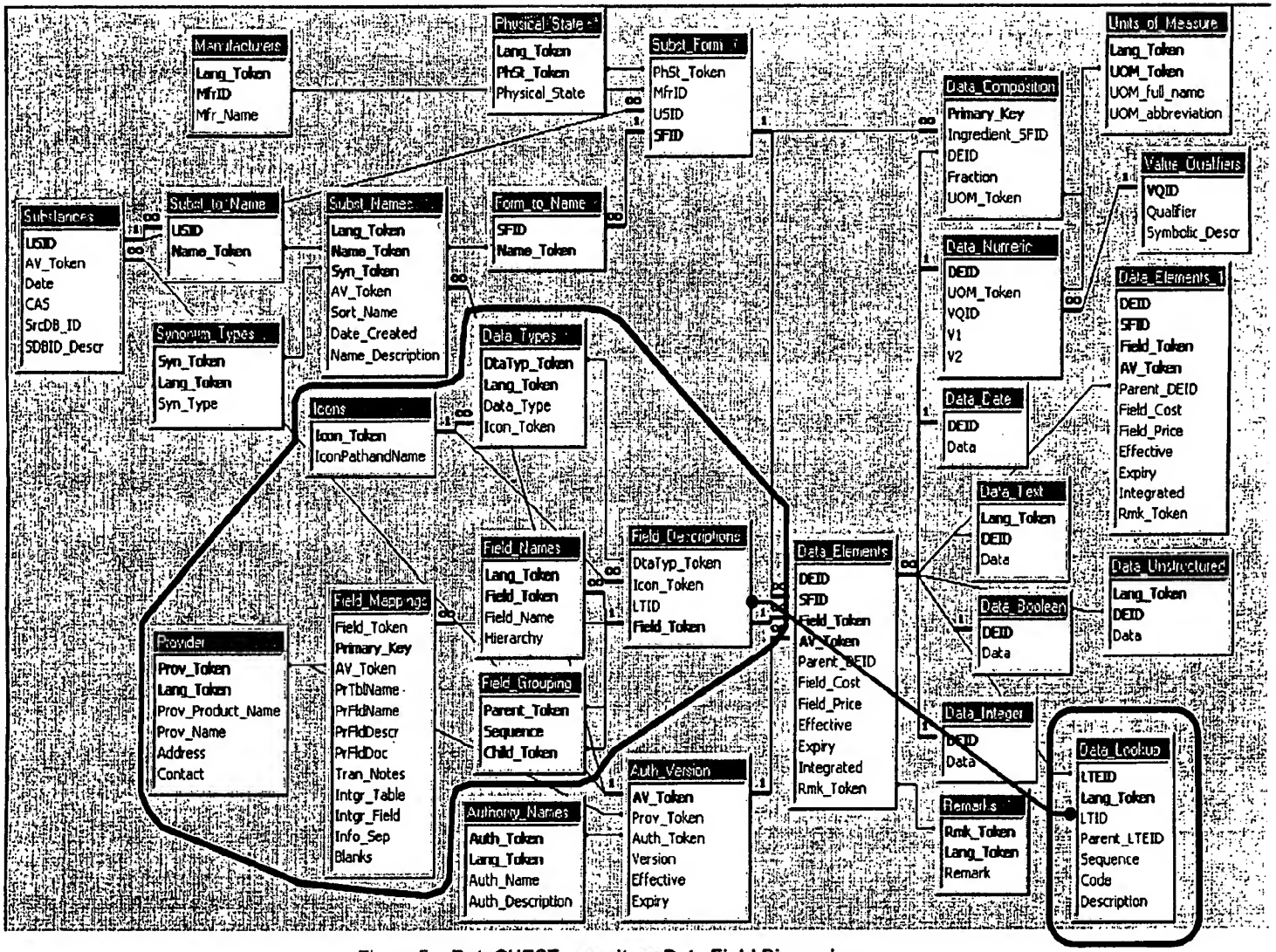


Figure 5 : **DataCHEST repository Data Field Dimension.**

The Authority/Version, or Source, Dimension

This last dimension of the Data Element table is the Authority/Version dimension. Each data element is identified by the source from which it has been provided (the Provider database), together with the original source authority (the Authority_Names database) from which the data was taken. Because data providers issue periodic releases of their data, versioning information must also be incorporated into this dimension. As well, it is not uncommon for multiple data providers to furnish data from the same source authority, as in some cases, this is the only valid way to obtain the pertinent data.

Title: The DataCHEST Database Schema

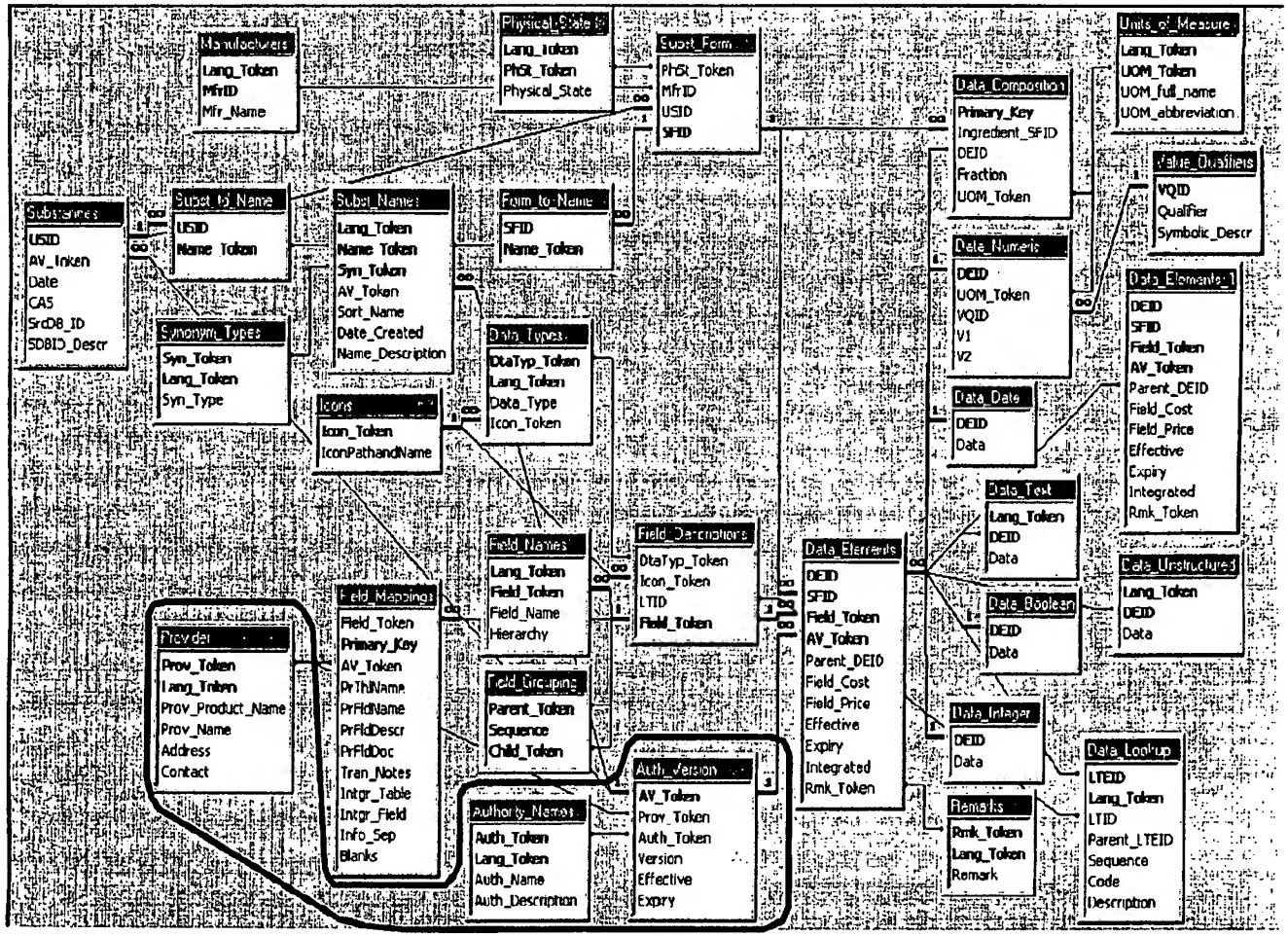
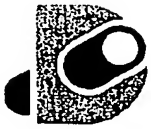


Figure 6 : **DataCHEST repository Authority/Version (Source) Dimension.**

083



The Data Dimension

As previously mentioned, data in the DataCHEST repository is stored in separate tables from the Data_Elements table, one table for each data type. Although technically not a dimensional element, we are considering it as such for the purposes of this document.

It is not intended that the table constructs remain static. Rather, new tables can be added as new data types are necessitated by the source databases that are integrated into the repository.

We have so far provided for the two basic data types:

- **Boolean**, for simple binary values, like "Yes"/"No", "True"/"False", "Present"/"Not Present", etc.
- **Integer**, for simple numeric values that are unitless, like year, count, etc., as well as for index entries into the Lookup-Table

As well, we have provided for multilingual text data types, consisting of a data element paired with a Language Token linking the data element to the Language table.

- **Text**, for multilingual text values up to the maximum allowable length for character strings under the lowest common denominator for ODBC databases, which is currently 255 characters.
- **Unstructured Text**, for multilingual text values that exceed the above maximum length for character strings.

Two additional data type constructs have been specially adapted to the source data that we have examined so far:

- **Numeric**, for data elements that are represented by non-integer values and/or that require a unit of measure for interpretation. This construct has a numeric data element and a link to a Units_of_Measure table, containing multilingual definitions of the various units of measure that have been defined during the source data transformation process.
- **Substance Composition**, for data elements defining the ingredients of substances in the Substance database that are mixtures made up of other substances in the Substance databases. This construct is composed of sets of ingredients, their fractions and the corresponding units of measure.

As indicated earlier, data elements can be further characterized by individual remarks held in the Remarks table.

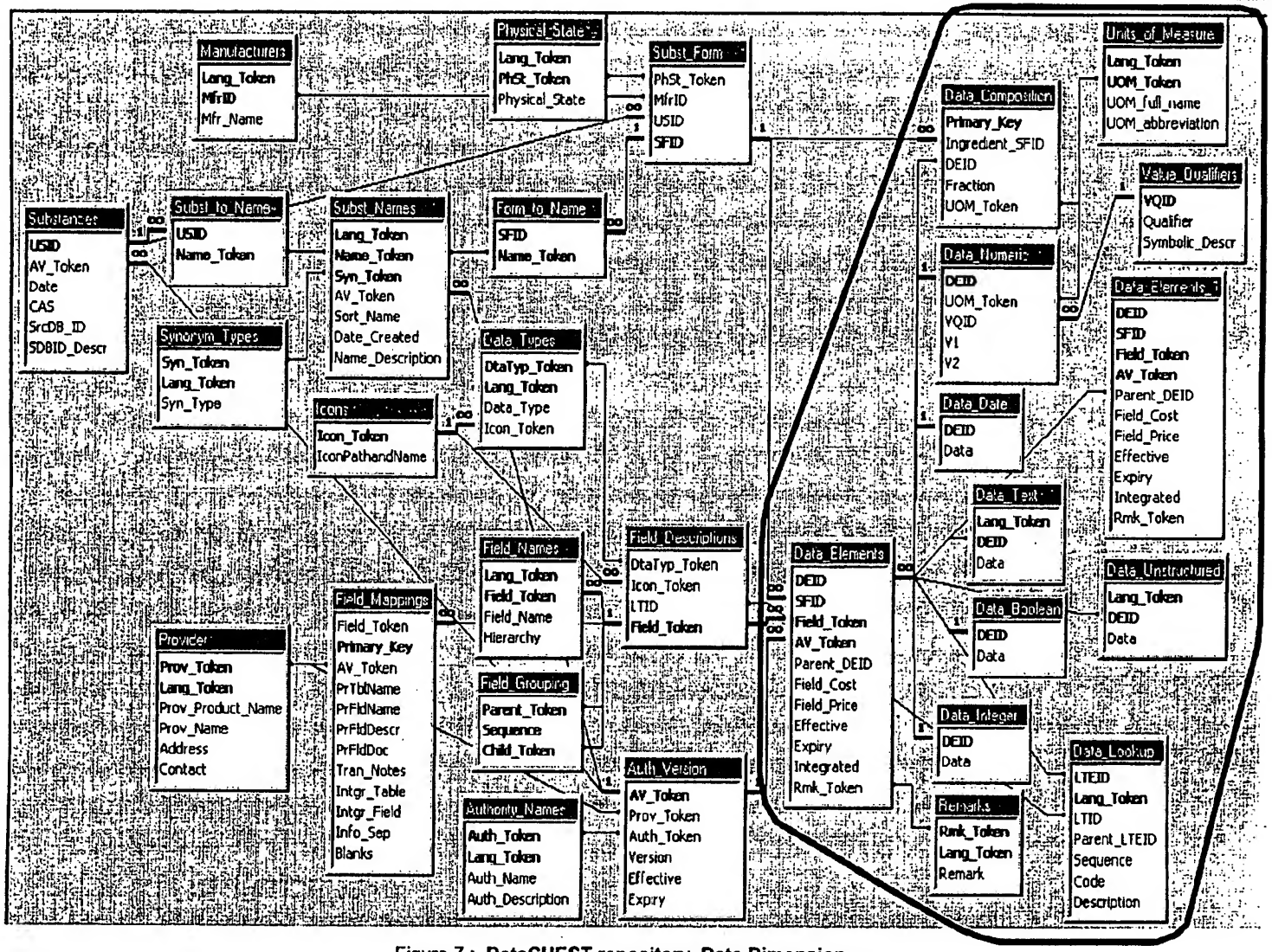


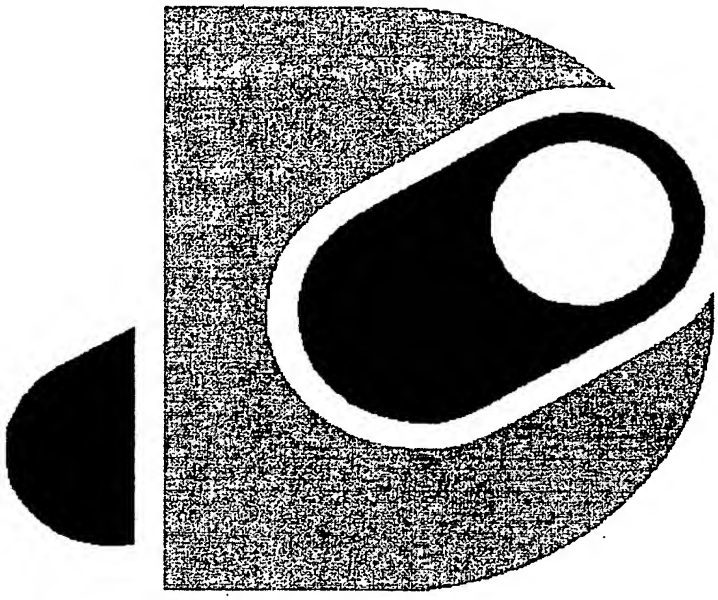
Figure 7 : DataCHEST repository, Data Dimension.

Index

Error! No index entries found.

BNSDOCID: <WO_____0159613A2_I_>

Title : Sub-Tree Concepts, Design and Implementation Strategies
Created : 2000-02-11 15:02 By : Alan S. Lawee
Last modified : 2000-02-11 15:40 By : Alan S. Lawee
Path & Filename : I:\CLIENTS-C-E (03-05)\0430-DATACHEST.COM, INC\0430-1-1P-PATENT APPLICATION\0430-1-1P APPLICATION\SUB-TREE CONCEPTS CLEAN.DOC



DataCHEST.com

Sub-Tree Concepts, Design and Implementation Strategies

Table of Contents

1 Abstract

This document discusses the **concepts** surrounding the creation and display of **Sub-Trees** in the DataCHEST Query Builder application, and offers implementation suggestions.

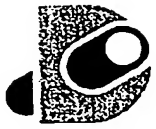
2 Concepts

The DataCHEST **Poly-Hierarchical Index Tree** is a unique and very powerful structure for classifying the various **Data Fields** contained in the **DataCHEST Repository**. At the same time as it provides an organizational reference ('table of contents') in which to locate a particular Data Field, it also provides a context for each node that can be used to help refine any searches that are performed for or on the node.

The large number (thousands) of **Data Fields** that are integrated into the **DataCHEST Repository** makes browsing the full tree a little cumbersome even though the tree is classified into topics and sub-topics. The concept of creating a sub-tree involves applying a variety of different filters against the entire tree-structure in order to be able to visualize a sub-set of the tree that may correspond to a particular set of constraints which better meet the criteria that a particular user is looking for.

For example, if a user is searching the **Index Tree** for data pertaining to a particular substance, it would be much more efficient to present an **Index Tree** containing only those data nodes and their respective parent topics which contain data for the substance under consideration. In another situation, if a user is looking for North American Occupational Exposure Limits, he or she will find them much more quickly if an **Index Tree** can be constructed from only those nodes that relate to both of these criteria.

It will become evident from the examples later in this document that the large variety of additional criteria that can be used to filter the **Index Tree** will increase the utility and usefulness of the **Index Tree** structure by several orders of magnitude.



3 Design

The full **Index Tree** is currently created off-line during the Data Repository update procedure by processing the tables in the Normalized Database instance with an Oracle PL/SQL script. This script creates a de-normalized table that maps to each node in the tree, even if the same field token appears more than once. The de-normalized table is then used to display the tree structure within the browser.

A similar process, written into a JSP (Java Server Page) routine can be used on-line to create a temporary table in exactly the same format in order to implement and display ad-hoc **Sub-Trees** at the user's discretion. The difference between the full **Index Tree** and the **Sub-Tree** is that the **Sub-Tree** is built using a subset of the nodes that appear in the full tree. The subset is determined by a variety of criteria that are meaningful to the user.

During the current design cycle, we will be implementing the following selection criteria for **Sub-Tree** creation:

1. Nodes that contain data pertaining to a set of substances selected by Name, CAS and/or other criteria
2. Nodes that contain data originating from a specific Provider
3. Nodes that contain data originating from a specific Authority
4. Nodes that contain data originating from a specific Provider Version
5. Nodes that contain data originating from a specific Authority Version
6. Nodes whose field name matches a keyword or topic search (e.g. contains the word 'drinking')
7. Nodes which contain one or more specific data types
8. Nodes found within a sub-branch of the full tree
   (These will usually also be found below other branches in the tree)

Note that the application should permit the user to create a **Sub-Tree** by combining one or more of the above criteria in a Boolean expression. Initially, the expression should be limited to a simple AND/OR combination of each criteria, but as the application becomes more sophisticated, complex Boolean expressions should be possible.

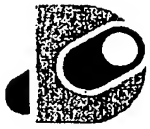
The details of each of the above **Sub-Tree Selection Criteria** will be discussed below.

3.1 Nodes that contain data pertaining to a substance or a set of substances

This criteria requires that a search be performed on the **Data_Elements** table for all records whose Substance Form Identifier (**SFID**) are found in the list resolved from the Substance Selection criteria specified by the User. The results must be grouped by the **Field_Token** column of the table, and then used to select which nodes are eligible for inclusion in the **Sub-Tree**.

A sample SQL request to extract the candidate **Field_Tokens** is listed below:

```
SELECT   DATA_ELEMENTS.FIELD_TOKEN, DATA_ELEMENTS.SFID
FROM     DATA_ELEMENTS
GROUP BY DATA_ELEMENTS.FIELD_TOKEN, DATA_ELEMENTS.SFID
HAVING   (((DATA_ELEMENTS.SFID) In (10346,11002)))
```



3.2 Nodes that contain data originating from a specific Provider

This next criterion requires that a search be again performed on the **Data_Elements** table, but this time for all records whose **Version_Token** is related to a specific **Provider** or a set of **Providers**. As before, the results must be grouped by the **Field_Token** column of the table, and then used to select which nodes are eligible for inclusion in the **Sub-Tree**.

The SQL request in this case is slightly more complex, as it must join several tables in order to extract the candidate **Field_Tokens**. A sample is again listed below:

```
SELECT   DATA_ELEMENTS.FIELD_TOKEN, PROVIDER_VERSIONS.PROV_TOKEN
FROM     DATA_ELEMENTS INNER JOIN (INFO_VERSIONS INNER JOIN
         PROVIDER_VERSIONS ON INFO_VERSIONS.PV_TOKEN =
         PROVIDER_VERSIONS.PV_TOKEN) ON DATA_ELEMENTS.VERSION_TOKEN =
         INFO_VERSIONS.VERSION_TOKEN
GROUP BY DATA_ELEMENTS.FIELD_TOKEN, PROVIDER_VERSIONS.PROV_TOKEN
HAVING   (((PROVIDER_VERSIONS.PROV_TOKEN)=1));
```

3.3 Nodes that contain data originating from a specific Authority

This next criterion requires that a search be again performed on the **Data_Elements** table, but this time for all records whose **Version_Token** is related to a specific **Authority** or a set of **Authorities**. As before, the results must be grouped by the **Field_Token** column of the table, and then used to select which nodes are eligible for inclusion in the **Sub-Tree**.

The SQL request in this case is a minor variation on the one used previously for the Provider criterion. A sample is again listed below:

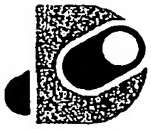
```
SELECT   DATA_ELEMENTS.FIELD_TOKEN, AUTH_VERSIONS.AUTH_TOKEN
FROM     (DATA_ELEMENTS INNER JOIN INFO_VERSIONS ON
         DATA_ELEMENTS.VERSION_TOKEN = INFO_VERSIONS.VERSION_TOKEN) INNER
         JOIN AUTH_VERSIONS ON INFO_VERSIONS.AV_TOKEN =
         AUTH_VERSIONS.AV_TOKEN
GROUP BY DATA_ELEMENTS.FIELD_TOKEN, AUTH_VERSIONS.AUTH_TOKEN
HAVING   (((AUTH_VERSIONS.AUTH_TOKEN)=1));
```

3.4 Nodes that contain data originating from a specific Provider Version

This next criterion requires that a search be again performed on the **Data_Elements** table, but this time for all records whose **Version_Token** is related to a specific **Provider Version** or a set of **Provider Versions**. As before, the results must be grouped by the **Field_Token** column of the table, and then used to select which nodes are eligible for inclusion in the **Sub-Tree**.

The SQL request in this case is a minor variation on the one used previously for the **Provider** criterion. A sample is again listed below:

```
SELECT   DATA_ELEMENTS.FIELD_TOKEN, INFO_VERSIONS.PV_TOKEN
FROM     (DATA_ELEMENTS INNER JOIN INFO_VERSIONS ON
         DATA_ELEMENTS.VERSION_TOKEN = INFO_VERSIONS.VERSION_TOKEN)
GROUP BY DATA_ELEMENTS.FIELD_TOKEN, INFO_VERSIONS.PV_TOKEN
HAVING   (((INFO_VERSIONS.PV_TOKEN)=1));
```



3.5 Nodes that contain data originating from a specific Authority Version

This next criterion requires that a search be again performed on the **Data_Elements** table, but this time for all records whose **Version_Token** is related to a specific **Authority Version** or a set of **Authority Versions**. As before, the results must be grouped by the **Field_Token** column of the table, and then used to select which nodes are eligible for inclusion in the **Sub-Tree**.

The SQL request in this case is a minor variation on the one used previously for the **Authority** criterion. A sample is again listed below:

```
SELECT   DATA_ELEMENTS.FIELD_TOKEN, AUTH_VERSIONS.AV_TOKEN
FROM     (DATA_ELEMENTS INNER JOIN INFO_VERSIONS ON
         DATA_ELEMENTS.VERSION_TOKEN = INFO_VERSIONS.VERSION_TOKEN)
GROUP BY DATA_ELEMENTS.FIELD_TOKEN, INFO_VERSIONS.AV_TOKEN
HAVING   (((INFO_VERSIONS.AUTH_TOKEN)=69));
```

3.6 Nodes whose field name matches a keyword or topic search

This criterion searches against the **Field_Names** table looking for matches against one or more keywords or keyword fragments. In the Visual Basic Prototype version, the results of this search were used to highlight matching tree nodes, but in the Browser-based Java version, it is much easier to simply create a sub-tree for the matches.

The sample SQL script is listed below:

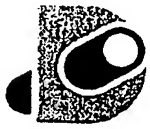
```
SELECT [FIELD_NAMES].[FIELD_TOKEN], LCase$([FIELD_NAME]) AS Expr1,
       [FIELD_NAMES].[LANG_TOKEN]
FROM FIELD_NAMES
WHERE (((LCase$([FIELD_NAME])) Like "%drinking%") And
       (([FIELD_NAMES].[LANG_TOKEN])=1));
```

3.7 Nodes which contain one or more specific data types

This criterion searches against the **Field_Descriptions** table looking for fields with one or more specific data types. This would be most useful for those users who are looking for specific information formats, for example, in order to load a database.

The sample SQL script is listed below:

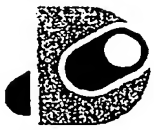
```
SELECT FIELD_DESCRIPTIONS.FIELD_TOKEN, FIELD_DESCRIPTIONS.DTATYP_TOKEN
FROM FIELD_DESCRIPTIONS
WHERE (((FIELD_DESCRIPTIONS.DTATYP_TOKEN) In (1,3,7,9,10)));
```



3.8 Nodes found within a sub-branch of the full tree

This is the original criterion for which the concept of the **Sub-Tree** was created. This feature allows the user to refine his search through the tree by specifying progressive refinements. For instance, creating a sub-tree on Regulatory Lists, followed by a second sub-tree on Regions – North America – United States – U.S. Federal, would yield a sub-tree containing only U.S. Federal Regulatory Lists, which could then be browsed by descending another branch, such as Health.

The easiest and quickest way to create this sub-tree would be to collect **Field Tokens** from the temporary GUI_DATA_INDEX table currently in use all the way down the branch from the currently-selected node, resolve any duplicate tokens, and then build a new table from the resulting list of field tokens. I will not hazard an attempt to create an SQL script for this task, as it is probably better handled by a programmed sequential read until the next peer or superior node is found.



4 Implementation Strategies

(NOTE: This preliminary strategy is only speculative and must be tested further. Also, it should be noted that both code examples included in the Appendices were written very quickly and can probably benefit considerably from some optimization analysis.)

In the design phase, samples of the SQL scripts that extract the list of candidate **Field Tokens** for the new **Sub-Tree** have been shown. The PL/SQL procedure script FIELD_DATA_INDEX that creates the GUI_DATA_INDEX table is included in Appendix A

The relevant **Sub-Trees** could be created by the existing FIELD_DATA_INDEX procedure by simply modifying the query inside the PL/SQL cursor **C_FIELDGRP**, declared in the **Get_Child** sub-procedure. A condition must be added to the **WHERE** clause indicating that **CHILD_TOKEN** is **IN** the set defined by the appropriate SQL statement for the method desired.

By way of example, to create a sub-tree for nodes containing the word "drinking", the **C_FIELDGRP** cursor could be defined as follows:

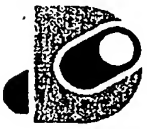
```
SELECT    CHILD_TOKEN, PARENT_TOKEN
FROM      FIELD_GROUPINGS
WHERE     PARENT_TOKEN = P_CHILD
AND       CHILD_TOKEN != PARENT_TOKEN
AND       CHILD_TOKEN IN
          (SELECT [FIELD_NAMES].[FIELD_TOKEN], LCase$([FIELD_NAME]) AS
                  Expr1, [FIELD_NAMES].[LANG_TOKEN]
          FROM      FIELD_NAMES
          WHERE     (((LCase$([FIELD_NAME])) Like "%drinking%") And
                    (([FIELD_NAMES].[LANG_TOKEN])=1))
          )
ORDER BY FIELD_GROUPINGS.DC_SEQUENCE
```

Once the **Sub-Tree** has been built, it must be 'cleaned' to remove field nodes without data and topic nodes without fields. This is accomplished using the logic described in Appendix B, which contains the commented Visual Basic code of a procedure written by the DataCHEST Integration team for this purpose.

Appendix A PL/SQL Procedure FIELD_DATA_INDEX

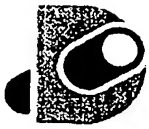
```
-- icit les rejets ne sont pas décompté dans le childcount voir si cela a un impacte

create or replace function FIELD_NAME_DATA_INDEXES (P_LANG_TOKEN number) return number is
--
-- Cette Fonction exécute le chargement de GUI_DATA_INDEXES
-- Si une erreur est détectée dans ce programme, alors le résultat
-- du renvoi de la fonction sera négatif.
--
-- Si aucune erreur: 0 est renvoyé.

-- Field_Descriptions

  v_FIELD_TOKEN field_descriptions.FIELD_TOKEN%type;
  v_LTID        field_descriptions.LTID%type;
  v_ICONS_TOKEN field_descriptions.ICONS_TOKEN%type;
  v_DTATYP_TOKEN field_descriptions.DTATYP_TOKEN%type;

-- Field_Names

  v_FIELD_NAME  field_names.FIELD_NAME%type;

-- Gui_Data_Indexes

  v_DATA_INDEX_ICON_NAME gui_data_indexes.DATA_INDEX_ICON_NAME%type;
  v_ARG_IS_TOKEN gui_data_indexes.ARG_IS_TOKEN%type;
  v_PARENT_DATA_INDEX_ID gui_data_indexes.PARENT_DATA_INDEX_ID%type;
  v_FILTER_SPEC_ID gui_data_indexes.FILTER_SPEC_ID%type;
  v_CHILDCOUNT gui_data_indexes.CHILDCOUNT%type;
  v_DATA_TYPE_ICON_NAME ICONS.ICONPATHANDNAME%type;

-- field_groupings

-- Autres
  v_ERROR integer;

  v_LEAF boolean;
  Erreur_Insertion exception;
  v_UNIQ_ID number(10);
-- icit enleve
   NBR_APPEL NUMBER(10);
```

```
function get_if_leaf return boolean is
begin
 if v_DTATYP_TOKEN is not null
  and v_DTATYP_TOKEN not in (12,2) then

    return (true);
 else
    return (false);

 end if;
end;


function get_ARG_IS_TOKEN return varchar2 is
begin
 if v_LEAF = TRUE then
   if (v_DTATYP_TOKEN = 9 or v_DTATYP_TOKEN = 1) then -- Si Lookups
     return('Y');
   else
     return('N');
   end if;
 else
   return ('');
 end if;
end;
```

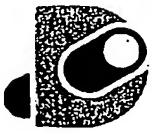


```
-- dataindexicon
function get_ICONPATHANDNAME (P_PARENT number) return varchar2 is

 v_TOKEN field_groupings.CHILD_TOKEN%type;
 v_ICONS icons.ICONS_TOKEN%type;

 cursor c_icons is
   select ICONPATHANDNAME
   from ICONS
   where ICONS_TOKEN = v_ICONS
   and ICONS_TOKEN != 0;


 cursor c_path is
   select gui.DATA_INDEX_ICON_NAME
   from gui_data_indexes GUI
   where gui.data_index_id = P_PARENT;

 v_ICONPATH ICONS.ICONPATHANDNAME%type;


begin

 v_ICONS := v_ICONS_TOKEN;

 open c_icons;
 fetch c_icons into v_ICONPATH;

 if c_icons%found then
   close c_icons;
   return (v_ICONPATH);
 else
   close c_icons;
   open c_path;
   fetch c_path into v_ICONPATH;
   close c_path;
 end if;

 return (v_ICONPATH);
end;


function get_LTID return number is
begin
 if (v_DTATYP_TOKEN = 9 or v_DTATYP_TOKEN = 1) then -- SI Lookups
   if v_LTID = 0 then
     return (-1);
   else
     return (v_LTID);
   end if;
 end if;

 return (to_number(''));
end;
```

```
-- datatypeicon
function get_DATA_TYPE_ICON_NAME return varchar2 is

 cursor c_icons is
   select ICO.ICONPATHANDNAME
   from ICONS ICO
     ,DATA_TYPES DTA
   where DTA.DTATYP_TOKEN = v_DTATYP_TOKEN
   and ICO.ICONS_TOKEN = DTA.ICONS_TOKEN;

 v_ICONPATH ICONS.ICONPATHANDNAME%type;

begin
 open c_icons;
 fetch c_icons into v_ICONPATH;

 if c_icons%found then
   close c_icons;
   return (v_ICONPATH);
 end if;

 close c_icons;

 return ('');
end;


function get_FILTER_SPEC_ID return number is

cursor c_filter is
  select FILTER_SPEC_ID
  from FILTER_SPECS
  where DTATYP_TOKEN = v_DTATYP_TOKEN;

  v_FILTER_SPEC_ID filter_specs.FILTER_SPEC_ID%type;

begin
 if v_LEAF = false then
   return (to_number(''));
 else
   if v_DTATYP_TOKEN is not null then
     open c_filter;
     fetch c_filter into v_FILTER_SPEC_ID;
     close c_filter;

     return (v_FILTER_SPEC_ID);
   else
     return (to_number(''));
   end if;
 end if;
end;
```

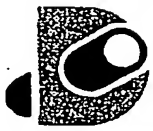


```
function get_CHILDCOUNT return number is

 cursor c_child is
   select count(CHILD_TOKEN)
   from FIELD_GROUPINGS
   where PARENT_TOKEN = v_FIELD_TOKEN;

   v_COMPTEUR integer;

begin
 if v_LEAF = true then
   return (to_number(''));
 else

   open c_child;
   fetch c_child into v_Compteur;
   close c_child;

   return (v_Compteur);

 end if;
end;
```



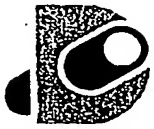
```
--
-- Child insertion in GUI_DATA_LOOKUPS table
--
Procedure Insert_Child (P_CHILD_TOKEN number, P_PARENT number) is

 v_NBR_INDENT    integer;

 --
 -- Récupération de l'information générale
 --
 cursor C_FIELD is
   select DESCR.FIELD_TOKEN
        ,DESCR.LTID
        ,DESCR.ICONS_TOKEN
        ,DESCR.DTATYP_TOKEN
        ,NAMES.FIELD_NAME
   from FIELD_NAMES NAMES
      ,FIELD_DESCRIPTIONS DESCR
   where NAMES.FIELD_TOKEN = DESCR.FIELD_TOKEN
   and   NAMES.LANG_TOKEN  = P_LANG_TOKEN
   and   DESCR.FIELD_TOKEN = P_CHILD_TOKEN;

  -- RECORD WITH FIELD TOKEN 0 IGNORED
  -- ENREGISTREMENT COMPORTANT UN TOKEN 0 IGNORÉ.

begin
 --
 -- Retrouver l'information de la table FIELD_DESCRIPTIONS
 --

 open  C_FIELD;
 fetch C_FIELD into v_FIELD_TOKEN
              ,v_LTID
              ,v_ICONS_TOKEN
              ,v_DTATYP_TOKEN
              ,v_FIELD_NAME;


 if C_FIELD%found then

 -- Insertion dans GUI_DATA_INDEXES

     v_LEAF := get_if_leaf;

     v_ARG_IS_TOKEN := get_ARG_IS_TOKEN;

     if v_DTATYP_TOKEN = 2 then
       v_DATA_TYPE_ICON_NAME := '';
     else
       v_DATA_TYPE_ICON_NAME := get_DATA_TYPE_ICON_NAME;
     end if;

     v_LTID := get_LTID;
     v_PARENT_DATA_INDEX_ID := P_PARENT;
     v_DATA_INDEX_ICON_NAME := get_ICONPATHANDNAME (v_PARENT_DATA_INDEX_ID);
```

```
v_FILTER_SPEC_ID := get_FILTER_SPEC_ID;
v_CHILDCOUNT := get_CHILDCOUNT;

if nvl(v_LTID,0) != -1 and v_FIELD_TOKEN is not null then

  if v_PARENT_DATA_INDEX_ID = 0 then
    v_PARENT_DATA_INDEX_ID := to_number('');
  end if;

  if v_DTATYP_TOKEN = 2 then
    v_DTATYP_TOKEN := to_number('');
  end if;


 begin
  insert into GUI_DATA_INDEXES
   (
    DATA_INDEX_ID
   ,DATA_INDEX_NAME
   ,INDEX_NO
   ,PRESET_INDEX_FLAG
   ,DATA_INDEX_ICON_NAME
   ,DATA_TYPE_ICON_NAME
   ,PARENT_DATA_INDEX_ID
   ,CHILDCOUNT
   ,FILTER_SPEC_ID
   ,LTID
   ,DTATYP_TOKEN
   ,ARG_IS_TOKEN
   )
  values
   (
    v_UNIQ_ID
   ,v_FIELD_NAME
   ,v_FIELD_TOKEN
   ,'N'
   ,nvl(v_DATA_INDEX_ICON_NAME,'UNKNOWN.GIF')
   ,v_DATA_TYPE_ICON_NAME
   ,v_PARENT_DATA_INDEX_ID
   ,v_CHILDCOUNT
   ,v_FILTER_SPEC_ID
   ,v_LTID
   ,v_DTATYP_TOKEN
   ,v_ARG_IS_TOKEN
   );

 v_UNIQ_ID := v_UNIQ_ID + 1;

  EXCEPTION
    when others then
NULL;
 -- ICIT VOIR          raise Erreur_Insertion;
    end;
```



```
    end if;
  end if;
  close C_FIELD;
 end;
```

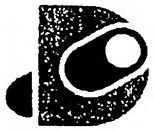
```
--
-- Search all Childs for all parents
--
Procedure Get_Child (P_CHILD number, P_PARENT_ID number)  is

  v_CHILD number(12);
  v_PARENT number(12);
  v_BOUCLE boolean;
  v_SEQTMP number(12);

  cursor c_FIELDGRP (P_CHILD number) is
    select CHILD_TOKEN, PARENT_TOKEN
      from FIELD_NAMES NAMES
         ,FIELD_DESCRIPTIONS DESCR
         ,FIELD_GROUPINGS GRP
      where PARENT_TOKEN = P_CHILD
      and CHILD_TOKEN != PARENT_TOKEN
      and NAMES.FIELD_TOKEN = DESCR.FIELD_TOKEN
      and NAMES.LANG_TOKEN  = P_LANG_TOKEN
      and DESCR.FIELD_TOKEN = GRP.CHILD_TOKEN
      order by GRP.DC_SEQUENCE;

 begin
-- icit
NBR_APPEL := NBR_APPEL + 1;

   v_BOUCLE := true;

--ICIT TEST ENLEVE EN BAS
-- IF v_UNIQ_ID > 3520 OR NBR_APPEL > 20 then
IF NBR_APPEL < 1 then
--   insert into sh_test values ('recursivite decouverte');
 NULL;
 ELSE

   open c_FIELDGRP (P_CHILD);
   while v_BOUCLE = true loop

       fetch c_FIELDGRP into v_CHILD, v_PARENT;

       if c_FIELDGRP%found then
         v_SEQTMP := v_UNIQ_ID;
         Insert_Child (v_CHILD, P_PARENT_ID);
         Get_Child (v_CHILD, v_SEQTMP);
       else
         v_BOUCLE := false;
       end if;

   end loop;
   close c_FIELDGRP;
END IF;

 end;
```



Main Code Body

```
begin
  delete from GUI_DATA_INDEXES
  where PRESET_INDEX_FLAG = 'N';
  commit;

  v_UNIQ_ID := 1;
-- ICIT ENLEVE
NBR_APPEL := 0;

  v_ERROR := 0;

  -- Child, Parent Id
  Get_Child (0, 0);

  commit;

-- Code d'erreur passé en paramètre
  v_ERROR := PRESET_DATA_INDEXES(1);

-- pas d'erreur
  return (v_ERROR);

  EXCEPTION
    when Erreur_Insertion then
        return (-10);
    when Others then
        return (-20);
end;
/
```

```
/* Test

declare
  aaa number(10);
begin
  delete from sh_test;
  aaa := FIELD_NAME_DATA_INDEXES (1);
  insert into sh_test values (aaa);
end;
/
select data_index_id, parent_data_index_id from gui_data_indexes order by 1;
SELECT COUNT(*) FROM GUI_DATA_INDEXES ;
select * from sh_test;

*/
```



Appendix B Code to remove nodes without any data

Pour supprimer tout les noeuds n'ayant aucune sortie vers des données, le processus doit s'effectuer en deux étapes :

1 - Supprimer tout les enregistrements de field_groupings dont le parent_token ou le child_token correspond a un élément de donnée (donc dont le dtatyp_token est contraire de 2) et dont le parent_token ou child_token ne retrouve sa correspondance dans data_elements sous le field_token.

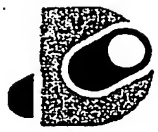
La requete :

```
Delete from field_groupings where parent_token in (select field_token from
field_descriptions where not(dtatyp_token=2) and field_token not in
(select field_token from data_elements group by field_token)) or
child_token  in (select field_token from field_descriptions where
not(dtatyp_token=2) and field_token not in (select field_token from
data_elements group by field_token))
```

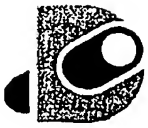
//**** A vérifier si on ne pourrait exécuter autrement pour éviter les full scan table "NOT IN"

2 - Retrouver à partir des root topics les bouts de branches
Verifier que ces bouts sont bien des éléments de données, sinon les supprimer
Retourner au parent des bouts et verifier s'ils ont d'autres enfants
Si oui, retrouver le bout de ces sous-branches et recommencer le processus ci-haut mentionné jusqu'à ce que tout les bouts soient des éléments de données (qui auront leur correspondance dans data_elements puisque la requete exécutée en premier lieu s'en est assuré)

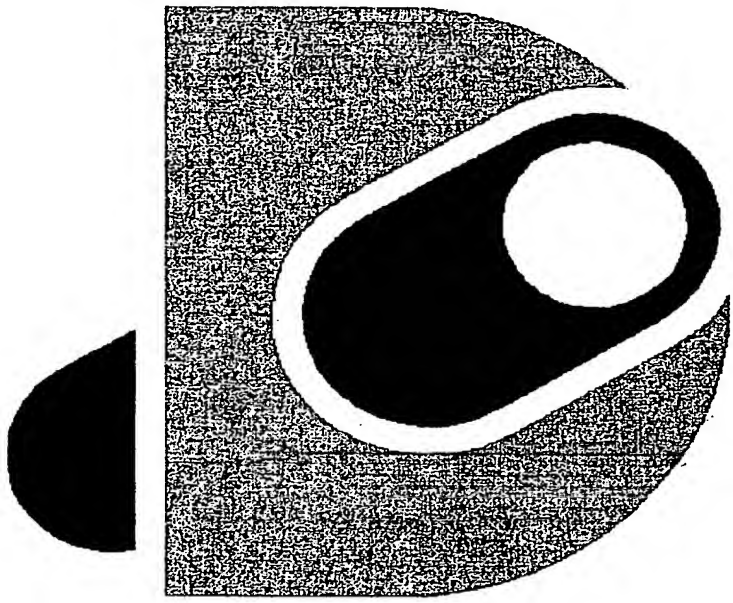
Le code :



```
Dim compteur As Long
Dim timer_jf As Long

Private Sub Clean_The_Tree()
    compteur = 1
    Dim db As Database
    Dim rc As Recordset
    Dim rc_data_type As Recordset
    Dim strSQL As String

    ****** All root topic from field_grouping
    strSQL = "SELECT Child_Token from Field_Grouping where Parent_Token = 0 order by DC_sequence"
    Set db = OpenDatabase("h:\chest.mdb")
    MsgBox strSQL
    Set rc = db.OpenRecordset(strSQL)
    While Not rc.EOF
        Call getChild(rc("Child_Token"), db)
        rc.MoveNext
    Wend
    rc.Close
    db.Close

    MsgBox "End"
End Sub
```



```
Sub getChild(ByVal strFieldToken As String, db As Database)
    Dim rc_grouping      As Recordset
    Dim rc_description   As Recordset
    Dim strSQL           As String
    Dim parent_token     As Long
    Dim rc               As Recordset

    strSQL = "SELECT Child_Token from field_grouping where parent_token = " & strFieldToken & " order by
    DC_sequence"
    Set rc = db.OpenRecordset(strSQL)
    '*** If there is no child for the current parent
    If rc.RecordCount = 0 Then
        '*** Retreive the data_type of this parent
        Set rc_description = db.OpenRecordset("select dtatyp_token from field_descriptions where field_token =" &
        strFieldToken)
        '*** If it's a structured object or a field_topic
        If rc_description("dtatyp_token") = 12 Or rc_description("dtatyp_token") = 2 Then
            '*** before to delete that node we have to find it's parents
            Set rc = db.OpenRecordset("select parent_token from field_grouping where child_token = " &
            strFieldToken)
            If Not rc.RecordCount = 0 Then
                parent_token = rc("parent_token")
                rc.Close
                '*** now we delete it
                db.Execute ("delete from field_grouping where child_token = " & strFieldToken)
                '*** now go get his parent to see if he still with some childs
                rc_description.Close
                Call getChild(CStr(parent_token), db)
            Else
                Exit Sub
            End If
        End If
        Else
        While Not rc.EOF
            Call getChild(rc("Child_Token"), db)
            DoEvents
            rc.MoveNext
        Wend
        rc.Close
    End If
End Sub

Private Sub Command1_Click()
    Clean_The_Tree
End Sub
```

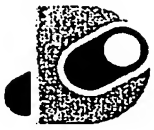


DataCHEST.com

DataCHEST Query Wizard
User Profiles

Table of Contents

BNSDOCID: <WO _ ___ _0159613A2 I_ >



Abstract

This document summarises the options and strategies for personalising the behaviour of the DataCHEST Query Wizard according to the preferences of the individual users.

User Profiles

Every user will develop his own profile to personalise the operation of the DataCHEST Query Wizard. The User Profile will define a number of different behaviour characteristics, which are discussed in this document.

Provider Priority

In the case where a query returns multiple values for the same data element, in other words, usually when more than one provider has supplied the same information, the User Profile will contain instructions on how to the user's preferences. A sample Query result, together with some possible alternatives, are listed below:

The query for the boiling point of ethylene glycol returns 3 values as listed below:

- *180° C* Price: $ 1.00 Provider A Royalty: $ 0.50
- *182° C* Price: $ 1.60 Provider B Royalty: $ 0.80
- *178° C* Price: $ 1.40 Provider C Royalty: $ 0.70

1. The Query will return the first value available from the most **preferred provider.** The User Profile can identify which providers are the most desirable, in order of priority, and the Query Wizard can return the appropriate result. If the provider preference is specified as alphabetic order, the query result would be :

   - *180° C* Price: $ 1.00 Provider A Royalty: $ 0.50

2. The Query should return a single value with the lowest price. In this case, the least costly data element will be returned. In the case where more than one data element have the same price, and it is the lowest, the Query Wizard will select the most preferred provider from the priority list described above. In this case the query result would again be :

   - *180° C* Price: $ 1.00 Provider A Royalty: $ 0.50

3. The Query should return all of the values. The user is not concerned with the usage charges and wants the most complete and corroborated information available. The query would cost the user $ 4.00, and the result and provider royalties would thus be :

   - *180° C* Provider A: Royalty: $ 0.50
   - *182° C* Provider B Royalty: $ 0.80
   - *178° C* Provider C Royalty: $ 0.70

4. The Query should return the calculated average of all of the values. In this case, the user should be charged somewhat less than the cost of retrieving all of the values, and the royalty charges should be divided among the different providers whose data elements were used, and this on a pro-rata basis according to their relative value of their royalty prices. A sample calculation follows:

   The value returned should be *"180° C based on an average of 3 data elements"*, the user should be charged $ 2.00 (for example), and the royalties of $ 1.00 (again for example) should be paid as follows:

   - Provider A Royalty: $ 0.25 (.5/2 *$1.00)
   - Provider B Royalty: $ 0.40 (.8/2 *$1.00)

Provider C Royalty: $ 0.35 (.7/2 *$1.00)

Many other forms of behaviour are possible, and will be implemented according to user demand.

Format of Result Set

The Query Result set can be returned in any of the following formats, according to the users' preference:

- HTML format, suitable for viewing in a browser
- One of a variety of Flat File formats, such as SDF (Standard Delimited Format), CSV (Comma-Separated Values), etc.
- ODBC format, suitable for direct storage into a users' local database, such as Microsoft Access or SQL Server, Oracle, Dbase, FoxPro, etc.

Other formats are not envisioned at this time, but will be evaluated if the need arises.

Language Preferences

The User Profile should contain the user's language preferences for

- **Retrieved Data.** The user can determine whether the Query Wizard will search for documents written in all languages or only in specified languages.
- **User Interface.** The user can select the language in which the User Interface is launched. Once launched, the user can switch between desired languages, but this parameter determines the language initially used..

Character Set Preferences

The User Profile should contain the user's character set preferences for

- **Retrieved Data.** The user can specify the character set to be used for the Result Set.

Launch Point

The User Profile should contain pointer information to automatically execute specific navigation and query commands upon launching to bring the user to a desired status within the Query Wizard. This should probably be implemented as a specially named 'start-up' macro.

Saved Requests

The User Profile should be designed to be able to contain a potentially unlimited set of saved requests. Using this feature, the users will be able to save requests and execute them on a repetitive basis. The implementation of a scheduler is contemplated for later implementation.

SQL Build strategy clean.doc 1

RETRIEVE request taken from the WebLogic Trace Log and edited for clarity

```
TRACE: call translateQuery(765, 1, 'AND', ?, ?, ?, ?, ?, ?)
translateQuery out parameters:

    resultRetrieve:
        SELECT
                OL_DATA_ELEMENTS.CAS "Substance CAS" ,
                SUBST_NAMES.NAME_DESCRIPTION "Substance Name" ,
                decode(
                        ol_data_elements.field_token,
                        '10091',

        GetDataText('DATA_TEXTS',1,'Varchar2',OL_DATA_ELEMENTS.DEID)
                        ,null
                ) "Molecular formula"

    resultFrom: FROM OL_DATA_ELEMENTS, SUBST_NAMES,SUBST_TO_NAMES

    resultWhere:
        WHERE  (        OL_DATA_ELEMENTS.USID = SUBST_TO_NAMES.USID
                AND     SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
                AND     SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
                AND     SUBST_NAMES.LANG_TOKEN= 1
                )
                AND     (       OL_DATA_ELEMENTS.CAS = '50000')
                AND     (       (OL_DATA_ELEMENTS.FIELD_TOKEN =10091) )

    resultOrderBy: ORDER BY   "Substance Name" ,orderbyNum("Substance CAS")

    insertFrom: FROM OL_DATA_ELEMENTS, SUBST_NAMES,SUBST_TO_NAMES
    insertWhere:
        WHERE  (        OL_DATA_ELEMENTS.USID = SUBST_TO_NAMES.USID
                AND     SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
                AND     SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
                AND     SUBST_NAMES.LANG_TOKEN= 1
                )
                AND     (   ( (OL_DATA_ELEMENTS.FIELD_TOKEN =10091) )   )

TRACE: buildQuery():  id=765 orgUserId=21

TRACE: call buildQuerySummary
        (
        765,
        21,
        'SELECT         OL_DATA_ELEMENTS.CAS "Substance CAS" ,
                SUBST_NAMES.NAME_DESCRIPTION "Substance Name" ,
                decode(
                        ol_data_elements.field_token,
                        '10091',

        GetDataText('DATA_TEXTS',1,'Varchar2',OL_DATA_ELEMENTS.DEID)
                        ,null
                ) "Molecular formula"',
        'FROM OL_DATA_ELEMENTS, SUBST_NAMES,SUBST_TO_NAMES',
        'WHERE          (       OL_DATA_ELEMENTS.USID = SUBST_TO_NAMES.USID
                AND     SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
                AND     SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
                AND     SUBST_NAMES.LANG_TOKEN= 1
                )
                AND     ( OL_DATA_ELEMENTS.CAS = '50000')
                AND     ( (OL_DATA_ELEMENTS.FIELD_TOKEN =10091))',
        'ORDER BY "Substance Name" , orderbyNum("Substance CAS")',
```

```
'FROM OL_DATA_ELEMENTS, SUBST_NAMES,SUBST_TO_NAMES',
'WHERE        (      OL_DATA_ELEMENTS.USID = SUBST_TO_NAMES.USID
       AND    SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
       AND    SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
       AND    SUBST_NAMES.LANG_TOKEN= 1
       )
       AND    (   ( (OL_DATA_ELEMENTS.FIELD_TOKEN =10091))  )'
)
```

Request as built by Application, extracted from WebLogic Trace log

```
SELECT OL_DATA_ELEMENTS.CAS "Substance CAS", SUBST_NAMES.NAME_DESCRIPTION
"Substance Name",
decode(ol_data_elements.field_token, '10091',
GetDataText('DATA_TEXTS',1,'Varchar2',OL_DATA_ELEMENTS.DEID), null ) "Molecular
formula"
FROM OL_DATA_ELEMENTS, SUBST_NAMES,SUBST_TO_NAMES
WHERE ( OL_DATA_ELEMENTS.USID = SUBST_TO_NAMES.USID
AND SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
AND SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
AND SUBST_NAMES.LANG_TOKEN= 1 )
AND ( OL_DATA_ELEMENTS.CAS = '50000')
AND ( (OL_DATA_ELEMENTS.FIELD_TOKEN =10091))
ORDER BY "Substance Name" , orderbyNum("Substance CAS")
```

... Takes forever.

I began this request in an Oracle SQLplus window, opened a second window, ran the segmented requests, formatted the results into this Word document and added my comments. I am now ready to print this document, and this request is STILL running.

Segmented request, part 1: Get USIDs and DEIDs

```
SELECT USID, DEID from OL_DATA_ELEMENTS
WHERE OL_DATA_ELEMENTS.CAS = '50000' and
OL_DATA_ELEMENTS.FIELD_TOKEN = 10091
```

... comes back quickly with 16 entries

```
     USID       DEID
--------- ---------
    11339   11023156
    11338   11023258
    21795   12167176
    21795   12495736
    21795   12481455
    21795   12377614
    21795   12377615
    21795   12377616
    21795   12377617
    21795   12377618
    21795   12377619
    21795   12377620
    21795   12296642
    21795   12506337
    21795   12232701
    21795   12449989
```

Segmented request, part 2: Get requested data

```
Select DC_DATA from DATA_TEXTS
WHERE DEID In (11023156, 11023258, 12167176, 12495736, 12481455, 12377614,
12377615, 12377616, 12377617, 12377618, 12377619, 12377620, 12296642,
12506337, 12232701, 12449989)
```

... comes back quickly with 16 entries

```
DC_DATA
----------------
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
CH2O
C31H31ClFN11O3S
CH2O
C12H24N2O
CH2O
```

Segmented request, part 3: Get Substance Names

```
Select NAME_DESCRIPTION from SUBST_NAMES, SUBST_TO_NAMES
Where USID in (11339, 11338, 21795)
and   SUBST_TO_NAMES.NAME_TOKEN = SUBST_NAMES.NAME_TOKEN
AND   SUBST_TO_NAMES.LANG_TOKEN = SUBST_NAMES.LANG_TOKEN
```

... comes back quickly with 70 entries

```
NAME_DESCRIPTION
--------------------------------------------------
OXOMETHANE
PARAFORM
OXYMETHYLENE
POLYOXYMETHYLENE GLYCOLS
SOLUTIONS KNOWN AS FORMALIN
SUPERLYSOFORM
TETRAOXYMETHYLENE
TRIOXANE
FA
FANNOFORM
IVALON
FORMALDEHYDE ... %
FORMALITH
FORMALDEHYDE, SOLUTIONS, WITH NOT LESS THAN 25 PER
FORMALINE (GERMAN)
FORMALINA (ITALIAN)
FORMALDEHYDE
FORMALDEHYD (CZECH, POLISH)
FORMALDEHYDE, AS FORMALIN SOLUTION (DOT)
FYDE
HOCH
KARSAN
FORMALDEHYDE (GAS)
FORMALDEHYDE, SOLUTIONS, FLAMMABLE
FORMALIN-LOESUNGEN (GERMAN)
FORMALIN (AS FORMALDEHYDE)
FORMIC ALDEHYDE
FORMALIN
FORMALIN 40
FORMOL
LYSOFORM
MORBICID
METHANAL
METHYLENE OXIDE
METHYLENE GLYCOL
METHYL ALDEHYDE
OPLOSSINGEN (DUTCH)
NCI-C02799
ALDEIDE FORMICA (ITALIAN)
ALDEHYDE FORMIQUE (FRENCH)
BFV
FYDE
FORMALDEHYDE ... %
FORMALDEHYDE, SOLUTIONS, WITH NOT LESS THAN 25 PER
FORMALDEHYDE (GAS)
FORMALIN
FORMALDEHYDE SOLUTION (FORMALIN)
FORMALDEHYDE
FORMALDEHYDE, SOLUTIONS, FLAMMABLE
FORMALIN (AS FORMALDEHYDE)
FORMOL
```

SQL Build strategy clean.doc 5

```
FORMALITH
MORBICID
FORMALDEHYDE
Formaldehyde
formaldehyde
Formaldehyde (gas)
FORMALDEHYDE (GAS)
formaldehyde ... %
Formaldehyde, solutions
Formaldehyde, solutions, flammable
Formaldehyde; Formalin
Formalin
FORMALIN
FORMALIN (AS FORMALDEHYDE)
Formic aldehyde
Methaldehyde
Methanal
Oxomethane
Oxymethylene
```

CLAIMS

Claimed is:

1. A process for integrating data from a variety of databases each with its own content, organization and structure into a single repository, whereby the user can then retrieve and display the integrated data in its original form, comprising:

a. transferring data from a source medium into a staging area;

b. converting said data into a database format;

c. mapping said data into a master Index Tree;

d. normalizing or de-normalizing at least a portion of said data into a format suitable for parsing and integration into a target database;

e. parsing said data to extract granular data;

f. exporting said data of step e. into said target database;

g. formatting said data in a normalized form;

h. verifying said target database to ensure integrity has been maintained and reproduction has been accurate;

i. archiving said target database to a storage media;

j. merging said data into a master database repository;

k. exporting said data from step j. to a pre-production instance; and,

l. de-normalizing said data from step k. so as to optimize access time in an online environment.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 August 2001 (16.08.2001)

PCT

(10) International Publication Number
WO 2001/059613 A3

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number:
PCT/IB2001/000369

(22) International Filing Date: 9 February 2001 (09.02.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/182,070 11 February 2000 (11.02.2000) US

(71) Applicant *(for all designated States except US)*: **DATACHEST.COM, INC.** [CA/CA]; 442 St. Gabriel Street, Suite 100, Montreal, Quebec H2Y 2Z9 (CA).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **LAWEE, Alan** [CA/CA]; 272 Harrow Crescent, Hampstead, Quebec H3X 3X6 (CA).

(74) Agent: **MBM & CO.**; Box 809, Station B, Ottawa, Ontario K1P 5P9 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
31 December 2003

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SYSTEM FOR DATA MANAGEMENT

(57) Abstract: A system and method for classifying, organizing and integrating raw information of all types into a form that permits the retrieval and analysis of the information by means of a simple user interface. Once integrated into the database and classified into the polyhierarchical tree structure, it is possible to request information using relationship that did not previously exist in the source data.

WO 2001/059613 A3

INTERNATIONAL SEARCH REPORT

Interna cation No
PCT/IB 01/00369

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, IBM-TDB, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 937 409 A (WETHERBEE JONATHAN) 10 August 1999 (1999-08-10) column 2, line 61 -column 3, line 40 column 4, line 33 -column 5, line 63 column 6, line 19-61 ---	1
A	VOOREN VAN L: "XML and legacy data conversion: introducing consumable documents" SGML EUROPE. CONFERENCE PROCEEDINGS, PROCEEDINGS OF SGML. THE NEXT DECADE - PUSHING THE ENVELOPE, XX, XX, 13 May 1997 (1997-05-13), pages 185-187, XP002166806 the whole document --- -/--	1

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 October 2003	31/10/2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Lázaro, M.L.

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	NITTEL S ET AL: "geoPOM: a heterogeneous geoscientific persistent object system" SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT, 1997. PROCEEDINGS., NINTH INTERNATIONAL CONFERENCE ON OLYMPIA, WA, USA 11-13 AUG. 1997, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 11 August 1997 (1997-08-11), pages 252-263, XP010245178 ISBN: 0-8186-7952-2 page 252, right-hand column, line 21 -page 253, right-hand column, line 40 page 254, left-hand column, line 40 -page 255, left-hand column, line 22	1

INTERNATIONAL SEARCH REPORT

Interna cation No
PCT/IB 01/00369

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5937409 A	10-08-1999	NONE	